POV-Ray Team

POV-Ray (Persistence of Vision Raytracer) versione 3.01 Manuale di Utilizzo

POV-Ray (Persistence of Vision Raytracer)

versione 3.01 - Manuale di Utilizzo

Traduzione Italiana a cura del Gruppo di Lavoro "POV.it"

Sommario

1. Introduzione

Questo manuale tratta in modo dettagliato l'uso del motore di raytracing Persistence of Vision™ (POV-Ray™). E' diviso in quattro parti : la guida all'installazione, un dettagliato tutorial, la guida al linguaggio di descrizione delle scene e l'appendice. La prima parte (vedi i capitoli "Descrizione del Programma" e "Iniziare Subito") spiega dove procurarsi e come installare POV-Ray e fornisce una breve introduzione al raytracing. Il tutorial spiega passo dopo passo come impiegare le differenti funzioni di POV-Ray (vedi il capitolo "Tutorial"). La guida al linguaggio fornisce una descrizione completa di tutte le funzioni di POV-Ray spiegando tutte le opzioni disponibili (che vengono impostate per mezzo della linea di comando o tramite parole chiave contenute nei file .INI) e il linguaggio di descrizione delle scene (vedi capitoli "Guida al Linguaggio di POV-Ray ", "Opzioni di POV-Ray" e "Linguaggio di Descrizione delle Scene").
L'appendice contiene alcuni suggerimenti, bibliografia, indirizzi per contattare gli autori e informazioni legali.
POV-Ray è basato su DKBTrace 2.12 di David K. Buck e Aaron A. Collins.

1.1 Notazione

Attraverso tutto il manuale, verrà usata la seguente notazione per distinguere le parole chiave del linguaggio di descrizione delle scene, i parametri della riga di comando, le parole chiave dei file INI (di inizializzazione) e i nomi di file.

`nome` parole chiave del linguaggio di descrizione
`nome` opzioni della riga di comando
`nome` parole chiave dei file INI (di inizializzazione)
nome nomi di file
nome indirizzo Internet, newsgroup Usenet.
Nota : nella versione ASCII del manuale non c'è differenza tra le varie notazioni.

2. Descrizione del programma

Il 'motore di raytracing' Persistence of Vision crea immagini tridimensionali di realismo fotografico utilizzando una tecnica di rendering detta raytracing. Il programma legge un file di testo che contiene informazioni che descrivono gli oggetti e l'illuminazione in una scena e genera un'immagine di quella scena dal punto di osservazione di una 'macchina fotografica' descritta nello stesso file. Il raytracing non è una tecnica veloce, ma produce immagini di altissima qualità con effetti realistici di riflessione, ombreggiatura, prospettiva e molto altro.

2.1 Cos'è il Raytracing ?

Il raytracing è una tecnica di resa (da qui in poi, rendering) che calcola un'immagine di una scena lanciando raggi di luce nella scena stessa. Per 'scena' si intende un insieme (finito, N.d.T.) costituito da oggetti, sorgenti luminose, dal punto di osservazione e da eventuali effetti speciali.
Per ogni punto dell'immagine finale, vengono tracciati nella scena uno o più raggi 'visivi' e viene controllata, per ogni raggio, l'intersezione con ogni oggetto che si trova nella scena. I 'raggi visivi' hanno origine nel punto di osservazione, rappresentato da una macchina fotografica (camera) e passano attraverso la 'finestra di osservazione' che rappresenta l'immagine finale.
Ogni volta che un oggetto è colpito da un raggio, viene calcolato il colore della sua superficie in quel punto. Per questo scopo si determina l'ammontare della luce proveniente da tutte le sorgenti luminose, per determinare se *quel* punto di quell'oggetto è in ombra, oppure no. Se la superficie dell'oggetto è riflettente, oppure traslucida, vengono impostati e tracciati nuovi raggi per determinare il contributo delle luci riflessa e rifratta al colore finale della superficie.
Caratteristiche speciali come la riflessione interdiffusa (radiosity) effetti atmosferici ed area lights,

ovvero luci estese su di una superficie, rendono necessaria la creazione di un grande numero di raggi supplementari all'interno della scena per ogni punto dell'immagine.

2.2 Cos'è POV-Ray ?

Il raytracer Persistence of Vision™ è stato sviluppato a partire dal programma DKBTrace 2.12 (scritto da David K. Buck e Aaron A. Collins) da un gruppo di persone, chiamato POV-Team™, nel loro tempo libero. Il quartier generale del POV-Team™ si trova nel forum POVRAY su CompuServe (vedi il paragrafo "Forum POVRAY su CompuServe", per maggiori informazioni).
Il pacchetto software contiene dettagliate istruzioni sull'uso del programma e sulla creazione di scene. Molte sorprendenti scene sono incluse con POV-Ray così da poter cominciare immediatamente a creare immagini. Queste scene possono essere modificate.
In aggiunta alle scene predefinite, sono fornite ampie librerie di oggetti geometrici e materiali predefiniti. E' possibile includere questi oggetti e materiali nella propria scena semplicemente includendovi il nome dell'oggetto (o del materiale) ed il nome del relativo file.
Ecco alcune funzioni di POV-Ray :

Semplice linguaggio di descrizione delle scene.
Vasta libreria di incredibili scene di esempio.
File da includere che predefiniscono molte forme geometriche, colori e materiali.
File immagine di output di altissima qualità (fino a 48 bit/pixel).
Display a colori a 15 e 24 bit su Personal Computer IBM (e compatibili) utilizzando hardware video appropriato.
Possibilità di creare paesaggi utilizzando campi di quota (height fields)
Luci coniche, cilindriche e puntiformi per sofisticati effetti di illuminazione.
Riflessione speculare e Phong per un maggiore realismo delle superfici.
Riflessione interdiffusa (radiosity) per un maggiore realismo dell'illuminazione.
Effetti atmosferici come nebbie, arcobaleni ed atmosfera.
'Aloni' (oggetti particellari) per modellare effetti come nuvole, polvere, fuoco e vapore.
Diversi formati di output per le immagini prodotte, tra cui Targa, PPM, PNG.
Primitive geometriche di base quali sfere, parallelepipedi, superfici quadriche, cilindri, coni, triangoli e piani.
Primitive geometriche avanzate quali toroidi, superfici di Bezier, campi di quota (montagne) blob, quartiche, triangoli, testo, frattali, superellissoidi, superfici di rotazione, prismi, poligoni, solidi di rotazione.
Le primitive geometriche possono essere facilmente combinate fra loro per formare oggetti più complessi mediante Geometria Costruttiva Solida o booleana (CSG, Constructive Solid Geometry). POV-Ray supporta unione, intersezione, differenza, fusione (merge).
Agli oggetti sono assegnati materiali chiamati texture (una texture descrive le proprietà di colore e di superficie di un oggetto).
Pigmentazioni e normali incorporate : Agate, Bozo, Bumps, Checker, Crackle, Dents, Granite, Gradient, Hexagon, Leopard, Mandel, Marble, Onion, Quilted, Ripples, Spotted, Spiral, Radial, Waves, Wood, Wrinkles e mappatura tramite immagini.
L'utente di POV-Ray può crearsi le proprie texture, od utilizzarne di predefinite come Ottone, Cromo, Rame, Oro Argento, Pietra, Legno.
Possibilità di creare texture combinate utilizzando strati di texture semitrasparenti o motivi modulari (tiles) o file di mappatura dei materiali.
Possibilità di mostrare un'anteprima dell'immagine mentre viene calcolata (non disponibile su tutte le piattaforme).
Possibilità di fermare il rendering quando è parzialmente completato e riprenderlo in seguito.

2.3 Quale versione di POV-Ray usare ?

POV-Ray può essere utilizzato sotto MS-Dos, Windows 3.x, Windows per Workgroups 3.11, Windows 95, Windows NT, Macintosh della serie 68000, Power PC, Commmodore Amiga, Linux, Unix ed altre piattaforme.
Le ultime versioni dei file necessari per l'esecuzione di POV-Ray sono disponibili su CompuServe, America On Line e diverse BBS. Vedi il paragrafo "Dove trovare i File di POV-Ray" per maggiori informazioni.

2.3.1 Personal Computer IBM e Compatibili

Attualmente ci sono tre diverse versioni per Personal Computer IBM che funzionano sotto diversi sistemi operativi (MS-Dos, Windows e Linux), come descritto più avanti.

2.3.1.1 MS-Dos

La versione MS-Dos gira sotto MS-Dos o come un'applicazione DOS sotto Windows 95, Windows NT, Windows 3.1 o Windows for Workgroups 3.11. Gira anche sotto OS/2 ed OS/2 Warp.

Requisiti Hardware e Software :

Processore 386 o superiore e almeno 4 megabyte di RAM.

Circa 6 megabyte di spazio su disco per l'installazione e 2-10 o più megabyte aggiuntivi come spazio di lavoro.

Un editor di testo capace di manipolare file ASCII. Il programma EDIT fornito con MS-Dos va bene per file di dimensione limitata (in dipendenza dalla RAM disponibile, N.d.T.).

Un programma per la visualizzazione di file immagine nei formati GIF e meglio ancora, TGA e PNG.

File di POV-Ray richiesti :

povmsdos.exe un archivio auto-estraente contenente il programma, le scene di esempio, i file 'include' standard e la documentazione sotto forma di file di aiuto ipertestuale con visore. Questo file si può anche trovare diviso in pezzi più piccoli per semplicità di downloading. Controllare la directory del proprio sito FTP o della propria BBS per vedere se devono essere scaricati altri file.

Hardware raccomandato :

Processore Pentium, o 486, o coprocessore matematico per 386 o 486 sx.

4 megabyte di RAM, o più.

Video SVGA preferibilmente con interfaccia VESA e possibilità di mostrare immagini in high color o true color.

Opzionalmente, il codice sorgente di POV-Ray.

Non è necessario per il funzionamento del programma. Viene fornito per i curiosi e gli avventurosi.
povmsd_s.zip Il codice sorgente C per POV-Ray per MS-Dos. Contiene parti generiche e parti specifiche per MS-Dos. Non contiene scene campione, file 'include' e documentazione. E' perciò necessario procurarsi anche l'archivio contenente gli eseguibili.
Un compilatore C che possa creare applicazioni a 32-bit in modalità protetta. Sono supportati Watcom 10.5a, Borland 4.52 con DOS Power Pack e, con limitazioni sulle parti grafiche, DJGPP1.12maint4. Non è supportato il compilatore DJGPP 2.0.

2.3.1.2 Windows

La versione per Windows gira sotto Windows '95, Windows NT e sotto Windows 3.1 o Windows 3.11 se sono installate le estensioni a 32 bit Win32s. Gira inoltre sotto OS/2 Warp.

Requisiti Hardware e Software :

Processore 386 o superiore con almeno 8 megabyte di RAM.

Circa 12 megabyte di spazio su disco per l'installazione e 2-10 megabyte o più per spazio di lavoro.

File di POV-Ray richiesti :

File **povwin3.exe** - un file auto-estraente ed auto-installante contenente il programma, le scene di esempio, file 'include' e documentazione. Questo file può essere diviso in diversi file più piccoli per semplificare il downloading. Controllare la directory del proprio sito FTP o della propria BBS per controllare se sono necessari altri file.

Sono raccomandati :

Processore Pentium o 486 dx o coprocessore matematico per 386 o 486 sx.

8 megabyte o più di RAM.

Video SVGA, preferibilmente con possibilità di mostrare immagini in high color o true color (15, 16, 32 bit per pixel).

Opzionalmente, il codice sorgente di POV-Ray per Windows non è necessario per il funzionamento di POV-Ray. Viene fornito per i curiosi e gli avventurosi.
povwin_s.zip Il codice sorgente in C per POV-Ray per Windows. Contiene parti generiche e parti specifiche per Windows. Non contiene le scene d'esempio, i file 'include' e la documentazione. Pertanto è necessario procurarsi anche gli archivi contenenti l'eseguibile.
POV-Ray per Windows può essere compilato solo con compilatori C che creano applicazioni Windows a 32 bit. Sono supportati Watcom 10.5a, Borland 4.5.2/5.0.

2.3.1.3 Linux

Requisiti Hardware e Software :

Processore 386 o superiore e almeno 4 megabyte di RAM.

Circa 6 megabyte di spazio su disco per l'installazione e 2-10 o più megabyte aggiuntivi per spazio di lavoro.

Un editor di testo capace di editare file ASCII (ad esempio, **vi**, oppure **emacs**, o **xedit**).

Un kernel Linux recente (dal 1994 in poi) e supporto per i binari in formato ELF. POV-Ray per Linux non è in formato .OUT.

Librerie ELF **libc.so.5**, **libm.so.5** ed una o entrambe le **libX11.so.6** o **libvga.so.1**.

File di POV-Ray necessari :

povlinux.tgz oppure **povlinux.tar.gz** - archivio contenente un eseguibile ufficiale per entrambe le versioni SVGALib e X-Windows. Contiene inoltre le scene d'esempio, file 'include' standard e documentazione.

Hardware raccomandato :

Processore Pentium o 486 dx o coprocessore matematico per 386 o 486 sx.

8 megabyte o più di RAM.

Video SVGA, preferibilmente con possibilità di visualizzare immagini in high color o true color (15, 16, 32 bit per pixel).

Un programma per la visualizzazione di file grafici nei formati PPM, PNG, TGA.

Opzionale, il codice sorgente non è necessario per l'utilizzo di POV-Ray per Linux. E' fornito per i curiosi e per gli avventurosi.
povuni_s.tar.gz oppure **povuni_s.tgz** Il codice sorgente in C per POV-Ray per Linux contiene parti generiche e parti specifiche per Linux. Non contiene le scene d'esempio, i file 'include' e la documentazione, pertanto è necessario procurarsi anche gli archivi contenenti l'eseguibile.
Il compilatore GNU C e opzionalmente, gli X-include file e librerie e CONOSCENZA DI COME USARLE.
Sebbene sia fornito codice sorgente per sistemi Unix generici, non viene fornita assistenza tecnica su come compilare il programma.

2.3.2 Apple Macintosh

La versione Macintosh gira sotto Apple MacOS dalla versione 7.0 in su, su ogni Macintosh con processore 68020-030-040, con o senza coprocessore matematico, o su qualunque computer della serie Power Macintosh.

Requisiti Hardware e Software :

Processore 68020 o superiore senza unità in virgola mobile (serie LC o Performa o Centris) ed almeno 8 megabyte di RAM, oppure

Un processore 68020 o superiore CON unità in virgola mobile (serie Mac II o Quadra) ed almeno 8 megabyte di RAM, oppure

Qualunque computer Power Macintosh ed almeno 8 megabyte di RAM.

MacOS 7 o superiore e Quickdraw (colore).

Circa 6 megabyte di spazio disco per l'installazione e 2-10 megabyte aggiuntivi per spazio di lavoro.

Visore di file grafici con supporto per i formati Mac PICT, GIF e possibilmente TGA e PNG

File di POV-Ray richiesti :

povmacnf.sit oppure **povmacnf.sit.hqx** - archivio stuffit contenente l'applicazione per serie 68k senza FPU, file esempio, file include, documentazione (versione più lenta per Mac senza coprocessore matematico), oppure

povmac68.sit o **povmac68.sit.hqx** - archivio stuffit contenente la versione per Macintosh serie 68k *con* FPU, scene d'esempio, file 'include' standard e documentazione (versione più veloce), oppure

povpmac.sit o **povpmac.sit.hqx** archivio stuffit contenente la versione per Power Macintosh, scene d'esempio, file 'include' standard e documentazione.

Hardware raccomandato :

Processore 68030/33 o più veloce, con FPU, o qualsiasi Power Macintosh.

Preferibilmente video a colori ; vanno bene anche 256 colori, ma migliaia o milioni di colori sono meglio.

Opzionale : il codice sorgente non è necessario per il funzionamento di POV-Ray : viene fornito per i curiosi e gli avventurosi. POV-Ray per Macintosh può essere compilato utilizzando Apple MPW 3.3, Metrowerks Code Warrior 8 o Symantec 8.
povmacs.sit oppure **povmacs.sit.hqx** - il codice sorgente completo per POV-Ray per Macintosh. Contiene parti generiche e parti specifiche per Macintosh. Non contiene le scene d'esempio, i file 'include' e la documentazione, pertanto è necessario procurarsi anche gli archivi contenenti l'eseguibile.

2.3.3 Commodore Amiga

La versione per Amiga è disponibile in diverse varianti : 68000/68020 senza FPU (molto lenta, non raccomandata), 68020/68881(68882), 68030/68882 e 68040. Esistono anche due sotto-versioni, una con interfaccia a sola riga di comando e una con interfaccia grafica (richiede MUI 3.1). Tutte le versioni girano con OS 2.1 e superiori.

Requisiti Hardware e Software :

Almeno 4 megabyte di RAM.

Almeno 2 megabyte di spazio disco per i file necessari, più 5-20 megabyte o più per spazio di lavoro.

Un editor di testo ASCII, interfaccia grafica configurabile per lanciarlo.

Un visore di file grafici. POV-Ray ha output nei formati TGA, PPM, PNG. Vengono inclusi convertitori della distribuzione PPMBIN per convertire questi ai formati IFF ed ILBM.

File di POV-Ray richiesti :

povami.lha- archivio LHA contenente l'eseguibile, scene d'esempio, file 'include' standard e documentazione.

Hardware raccomandato :

8 megabyte o più di RAM.

Processore 68030 e 68882 o superiori.

Scheda video a 24 bit (è supportata la libreria CyberGFX)

Appena sarà disponibile un compilatore stabile per i sistemi Amiga Power PC, è in programma una versione per questa piattaforma.

Opzionale : il codice sorgente non è necessario per il funzionamento di POV-Ray : viene fornito per i curiosi e gli avventurosi.
povlha_s.zip - Il codice sorgente in C per Amiga. Contiene parti generiche e parti specifiche per Amiga. Non contiene le scene d'esempio, i file 'include' e la documentazione, pertanto è necessario procurarsi anche gli archivi contenenti l'eseguibile.

2.3.4 SunOS

Requisiti Hardware e Software :

Processore Sun SPARC ed almeno 4 megabyte di RAM.

Circa 6 megabyte di spazio disco per l'installazione ed almeno 2-10 megabyte aggiuntivi per spazio di lavoro.

Un editor di testo capace di editare file ASCII.

Sistema operativo SunOS 4.1.3 oppure un altro sistema operativo capace di eseguire un binario di questo genere (Solaris o Linux per SPARC).

File di POV-Ray necessari :

povsunos.tgz oppure **povsunos.tar.gz** - archivio contenente un binario eseguibile ufficiale per le modalità solo testo o X-Windows. Contiene anche scene d'esempio e file 'include', più la documentazione.

Hardware raccomandato :

8 megabyte di RAM, o più.

Se si desidera visualizzare le immagini su monitor, X-Windows oppure un X-Term.

Preferibilmente possibilità di video a 24 bit per quanto il codice per display sotto X funzioni con qualunque combinazione di risoluzione e colori.

Visore di file grafici capace di leggere i formati TGA, PNG, PPM

Opzionale : il codice sorgente non è necessario per il funzionamento di POV-Ray : viene fornito per i curiosi e gli avventurosi.
povuni_s.tgz oppure povuni_s.tar.gz - Il codice sorgente in C per Unix. Contiene parti generiche e parti specifiche per Unix. Non contiene le scene d'esempio, i file 'include' e la documentazione, pertanto è necessario procurarsi anche gli archivi contenenti l'eseguibile.
Un compilatore C e opzionalmente file 'include' e librerie X e conoscenza di come usarle.
Per quanto venga fornito il codice sorgente per sistemi Unix generici, non è data assistenza su come compilare il programma.

2.3.5 Sistemi Unix generici.

Software necessario :

povuni_s.tgz oppure povuni_s.tar.gz - Il codice sorgente C per Unix. Contiene parti generiche e parti specifiche per Unix, compreso X-Windows.

povuni_d.tgz oppure un qualunque archivio contenente le scene d'esempio, i file 'include' standard e la documentazione. Questo archivio potrebbe ad esempio essere quello per Linux o per SunOS descritto sopra.

Un compilatore C e conoscenza di COME USARLO. Per quanto venga fornito il codice sorgente per sistemi Unix generici, non è data assistenza su come compilare il programma.

Un editor di testo capace di editare testi ASCII (**vi**, **emacs**, **xedit**, ad esempio).

Hardware raccomandato :

Coprocessore matematico.

8 megabyte di RAM o più.

Visore di file grafici nei formati PPM, TGA o PNG.

Opzionali :
X-Windows qualora si vogliano vedere le immagini durante il rendering.
Gli 'include' file di X-Windows. Se non si è pratici nel compilare programmi per X-Windows può essere necessario l'aiuto di una persona esperta nell'installazione del sistema operativo, dato che le librerie di X-Windows non sono sempre nella stessa posizione.

2.3.6 Tutte le versioni

Ogni archivio eseguibile contiene la documentazione completa per POV-Ray insieme a istruzioni specifiche per il determinato tipo di piattaforma.
Tutte le versioni del programma dividono le stesse funzioni come oggetti, illuminazione e materiali. In altre parole, un Personal Computer può creare le stesse immagini di un supercalcolatore Cray, purché abbia abbastanza memoria (ed il proprietario abbastanza tempo, N.d.T.).
L'utente vorrà procurarsi l'eseguibile che meglio si adatta al proprio hardware. Vedi il paragrafo "Dove Trovare i File di POV-Ray" per indicazioni su dove reperire i file necessari. Queste fonti possono essere contattate per informazioni su quale sia la versione che meglio si adatta alle proprie esigenze ed alle proprie disponibilità hardware.

2.3.7 Compilare POV-Ray

(Questo paragrafo tende a scoraggiare in maniera sufficientemente terroristica la compilazione casalinga di POV-Ray, N.d.T.)
Le sezioni successive saranno di aiuto a compilare il codice C in cui è scritto POV-Ray in un eseguibile funzionante. Sono dirette solo a quelle persone che desiderino compilare una versione personalizzata di POV-Ray o portare il software su di una piattaforma non ufficialmente supportata.
La prima domanda da porsi è :"Ho realmente bisogno di compilare POV-Ray ?" Le versioni eseguibili rilasciate dal POV-Ray Team, sono disponibili per MS-Dos, Windows 3.1x/95/NT, Mac 68k, Mac Power PC, Amiga, Linux per processori Intel x86 e SunOS. Altre piattaforme possono essere supportate da versioni non ufficiali. Se non si intende aggiungere funzioni personalizzate o sperimentali al programma, o se esiste già un eseguibile per la piattaforma, allora non c'è bisogno di compilare questo programma.
Volendo proseguire, si dovrebbe tenere conto che si è pressoché da soli : le sezioni seguenti e altra documentazione collegata alla compilazione assume che si sappia cosa si sta facendo. Assume che si abbia un compilatore C adeguato, installato e funzionante, che si sappia come compilare e linkare grandi programmi a sezioni multiple, usando un'utilità MAKE o un project file IDE se la piattaforma lo supporta. Dato che i MAKEFILE e i project file generalmente specificano un drive, non si promette che i makefile o i projects distribuiti funzionino su di un determinato sistema. Si assume che si sappia come modificare un makefile e i projects per specificare dove si trovano le librerie del sistema e gli altri file necessari.
In generale, non dovrebbe essere atteso alcun aiuto dal POV-Ray Team su come compilare il programma. Tutto viene fornito "così com'è". Tutto ciò che si può dire con qualche certezza è che "noi siamo riusciti a compilarlo sui nostri sistemi. Se sul tuo non funziona, probabilmente non

sappiamo dirti perché".
Non c'è documentazione tecnica per il codice sorgente eccetto i commenti nei file. Il codice è scritto in maniera chiara ed è ben commentato, ma alcune sezioni sono praticamente appena commentate ed alcuni commenti potrebbero essere obsoleti. Non è fornita assistenza su come aggiungere funzioni al programma. Non viene spiegato come funziona una determinata caratteristica del programma. In certi casi, la persona che ha scritto una parte del programma, non è più attiva nel Team e non si sa esattamente come funzioni quella parte.
Quando si compilano versioni non ufficiali o personalizzate di POV-Ray, assicurarsi di avere letto e seguito tutte le norme della licenza di Copyright. In generale, si può modificare e usare POV-Ray personalmente in qualunque modo, ma se si distribuiscono versioni non ufficiali, devono venire seguite le regole dettate dal Team. Non si possono utilizzare, in alcuna circostanza, porzioni del codice sorgente di POV-Ray in altri programmi.

2.3.7.1 Struttura delle directory.

Il codice sorgente di POV-Ray è distribuito in archivi, dove i file sono disposti secondo una particolare gerarchia di directory (o cartelle, a seconda del sistema operativo). Quando si estraggono gli archivi, lo si dovrebbe fare in maniera che mantenga intatta la struttura delle directory (ad esempio, pkunzip -d oppure tar -xvfz). In generale, si suggerisce di creare una directory chiamata **povray3** ed estrarre i file al suo interno. L'estrazione creerà una directory chiamata **source** con molti file e sotto-directory. In generale ci sono archivi separati per ogni piattaforma hardware e sistema operativo, ma ciascuno di questi archivi può supportare più di un compilatore. Ad esempio, qui è riportata la struttura delle directory per l'archivio MS-Dos

source
source\libpng
source\zlib
source\msdos
source\msdos\pmode
source\msdos\borland
source\msdos\djgpp
source\msdos\watcom

La directory **source** contiene i file sorgenti per le parti generiche di POV-Ray, uguali su tutte le piattaforme. La directory **source\libpng** contiene file per la compilazione di una libreria di routine usate nel leggere e scrivere immagini in formato PNG (Portable Network Graphics). La directory **source\zlib** contiene file per la compilazione di una libreria utilizzata da *libpng* per comprimere e decomprimere flussi di dati. Tutti questi file sono utilizzati da tutte le piattaforme e tutti i compilatori e si trovano in ogni versione degli archivi contenenti i sorgenti.
La directory **source\msdos** contiene tutti i sorgenti per la versione MS-Dos comuni a tutti i compilatori supportati. La sub-directory **source\msdos\pmode** contiene i sorgenti per la libreria **pmode.lib** che è richiesta da tutte le versioni per MS-Dos. Le sottodirectory **borland**, **djgpp**, **watcom** contengono sorgenti, makefile e project file per i rispettivi compilatori C. Questa directory è unica nell'archivio per MS-Dos. Analogamente la versione per Windows contiene una directory **source\windows**, la versione Unix contiene source\unix ecc.
La directory **source\msdos** contiene un file, **cmpl_msd.doc**, nel quale si trovano informazioni relative alla compilazione specifiche per la versione MS-Dos. Altre versioni contengono file analoghi, di nome **cmpl_xxx.doc.** Assicurarsi di avere letto tutti i file **cmp_xxx.doc** pertinenti alla propria piattaforma e compilatore.

2.3.7.2 Configurare il codice sorgente di POV-Ray

Per ogni piattaforma si ha un file *header*, **config.h** che si trova generalmente nella directory specifica per la piattaforma, ma che si può anche trovare nella directory specifica per il compilatore. Alcune piattaforme hanno più versioni differenti di questo file, per cui può presentarsi la necessità di copiarlo e rinominarlo come **config.h**. Questo file è incluso in ogni modulo del programma. Contiene tutti i prototipi, le procedure (macro) o altre definizioni che sono necessarie per le parti generiche del programma, ma che devono essere messe a punto manualmente per una particolare piattaforma o compilatore.
Per esempio, diversi sistemi operativi utilizzano caratteri diversi come separatori tra i nomi delle directory e dei file. MS-Dos utilizza il 'back slash' (\), Unix il 'front slash' (/), MacOS i due punti. Il file **config.h** per MS-Dos e Windows contiene il testo seguente :

```
#define FILENAME_SEPARATOR '\'
```

Che istruisce la parte generica di POV-Ray ad usare un back slash.
Ogni personalizzazione di cui la parte generica del codice sorgente ha bisogno, ha una regolazione standard nel file **souce\frame.h** che è incluso in ogni modulo. L' header **frame.h** contiene molti gruppi di definizioni di questo tipo :

```
#ifndef FILENAME_SEPARATOR
#define FILENAME_SEPARATOR '/'
#endif
```

che significa 'se questo parametro non è stato definito prima in **config.h** allora questo è un valore di default'. Controllare quindi il file **frame.h** per vedere quali altri valori devono essere configurati. Se vengono utilizzate delle definizioni per specificare funzioni specifiche per una data piattaforma, deve essere incluso anche un prototipo per quella funzione. Il file **source\msdos\config.h** per esempio, non solo contiene la macro :

```
#define POV_DISPLAY_INIT(w,h) MSDOS_Display_Init ((w), (h));
```

per definire il nome della funzione di inizializzazione del display grafico, ma contiene anche il prototipo :

```
void MSDOS_Display_Init (int w, int h);
```

Progettando di portare POV-Ray ad una piattaforma non supportata, probabilmente è meglio cominciare con la più semplice versione (senza output a video) generica per Unix, quindi aggiungere nuove parti tramite il file **config.h**.

2.3.7.3 Conclusione

Capiamo che le sezioni sopra costituiscono solo i primi passi, ma metà del divertimento nel lavorare su POV-Ray è impararlo ed approfondirlo da soli. E' così che i membri del POV-Ray Team hanno iniziato. Abbiamo provato a scrivere il codice sorgente nel modo più chiaro possibile. Assicuratevi di leggere **cmpl_xxx.doc** nelle directory specifiche per la vostra piattaforma e per il vostro compilatore per ottenere aiuto di minore importanza se state lavorando su una piattaforma o computer non supportati dalle versioni ufficiali.
Buona Fortuna !

2.4 Dove trovare i File di POV-Ray

Le versioni più recenti di POV-Ray sono disponibili alle seguenti fonti.
2.4.1 Forum POVRAY su CompuServe
2.4.2 Internet
2.4.2.1 Siti FTP per POV-Ray raggiungibili dall’Italia

2.4.1 Forum POVRAY su CompuServe

Il quartier generale di POV-Ray è su CompuServe nel forum POVRAY, gestito da alcuni dei membri del team. Lì ci incontriamo per dividere informazioni, programmi, utilità e immagini create con POV-Ray. Chiunque è invitato ad unirsi al forum su CIS-POVRAY. Speriamo di incontrarvi là ! Potete trovare informazioni su come entrare in CompuServe chiamando (800)848-8990 o visitando la homepage di CompuServe *http://www.compuserve.com* . Accesso diretto a CompuServe è disponibile anche in Giappone, Europa e molti altri paesi.

2.4.2 Internet

La 'casa' su Internet di POV-Ray è raggiungibile su World Wide Web (WWW) all'indirizzo *http://www.povray.org* e tramite ftp come *ftp.povray.org.* Passateci spesso per gli ultimi file, utilities, notizie ed immagini del sito internet ufficiale di POV-Ray.
Il newsgroup *comp.graphics.rendering.raytracing* è popolato da molti utenti competenti di POV-Ray, desiderosi di condividere le loro conoscenze. In generale, chiedono di leggere qualche articolo per controllare se qualcuno abbia già risposto alla stessa domanda e, naturalmente, di seguire una corretta 'netiquette'. Se ci sono dubbi riguardo la loro qualificazione ed autorevolezza, alcuni minuti spesi alla Ray Tracing Competition a *www.irtc.org* vi convinceranno velocemente !

2.4.2.1 Siti FTP per POV-Ray raggiungibili dall'Italia.

Questi siti costituiscono dei 'mirror' del sito ufficiale *ftp.povray.org* e vengono aggiornati con una certa frequenza. La ricezione dati dall'Italia è veloce e permettono di scaricare mediante FTP anonimo tutti i file correlati a POV-Ray di cui si possa avere bisogno.

ftp-fh.wolfenbuettel.de/pub/graphics/povray/
ftp.flashnet.it/mirror2/ftp.povray.org/povray/
ftp.informatik.uni-oldenburg.de/pub2/pov-ray/

2.4.3 Area dedicata a PC-Graphics su America On-Line

C'è un'area su America On-Line dedicata a informazioni su POV-Ray. Si trova nella sezione PC-Graphics di AOL, con la parola chiave POV. Quest'area contiene anche gli eseguibili per Macintosh. E'meglio se i messaggi sono lasciati nella sezione 'Company Support'. Attualmente, Bill Pulver è il nostro rappresentante.

2.4.4 The Graphics Alternative BBS - El Cerrito, California

Per quelli che vivono sulla West Coast degli Stati Uniti, i file di POV-Ray si possono trovare sulla BBS 'The Graphics Alternative'. E' una grande BBS di grafica gestita da Adam Shiffman. TGA è un sistema di BBS di alta qualità, che offre servizi di messaggeria e file a oltre 1300 utenti.

510-524-2780 (PM14400FXSA v.32bis 14.4k, Pubblico)
Tel. 510-524-2165 (USR DS v.32bis/HST 14.4k, Associati)

2.4.5 PCGNet

Il 'Professional Cad and Graphics Network' (PCGNet) serve le comunità del CAD e della grafica rendendo ampiamente disponibili utili informazioni. Precedentemente chiamato ADENet, PCGNet è una nuova rete creata da zero, che incorpora nuovi nodi e si occupa sia di CAD che di argomenti

connessi alla grafica, tra cui progettazione, disegno, ingegneria, modellazione 2d e 3d, applicazioni multimediali, sistemi, immagini raster, rendering 3d ed animazione.
PCGNet è progettato per soddisfare le necessità di tutti coloro che chiamano stimolando interesse e producendo forum di supporto per gli utenti attivi che abbiano interesse negli argomenti precedentemente elencati ; interesse e supporto sono prodotti attraverso conferenze, condivisione di file attraverso la rete e novità e informazione dall'industria del software e dalla stampa. Le conferenze di PCGNet sono forum moderati ideati per ospitare discussioni informali, seppure professionali su materie collegate a CAD e grafica.

2.4.6 Libri e CD-ROM su POV-Ray

Quelli che seguono sono prodotti creati da membri del POV-Team. Nonostante siano aggiornati alla versione 2.2 del programma, sono ancora utili. Il CD-ROM verrà aggiornato alla versione 3.0, mentre è in preparazione una nuova versione del programma che sarà pronta nella prima metà del 1997. (la versione 3.01 è stata pubblicata il 7 febbraio, N.d.T.).
I libri sotto elencati sono stati recentemente catalogati come 'fuori stampa', ma possono ancora essere trovati in qualche libreria o biblioteca. (visitare *http://www.dnai.com :80/waite/* per maggiori dettagli).

Tracing Creations, 2d Ed.Young and Drew Wells 1-878739-69-7 Group Press 1994
pagine con inserto a colori e POV-Ray 2.2 su dischi MS-Dos da 3.5"

Tracing Worlds with POV-Ray
Alexander Enzmann, Lutz Kretzschmar, Chris Young,
1-878739-64-6
Group Press 1994
Contiene il modellatore Moray 1.5x e POV-Ray 2.2 su dischi MS-Dos da 3.5"

Tracing for the Macintosh CD
Eduard Schwan
1-878739-72-7
Group Press, 1994
Contiene un CD-ROM pieno di immagini, scene e filmati QuickTime ed una guida al linguaggio interattiva per parole chiave. Inoltre un floppy disk con POV-Ray per coloro che fossero sprovvisti di lettore CD-ROM.

Raytrace ! The official POV-Ray CD-ROM.
Il CD-ROM ufficiale di POV-Ray è una raccolta di immagini, scene, sorgenti per il programma, utilità e suggerimenti su POV-Ray e la grafica in 3D presi da Internet e CompuServe. Questo CD è destinato non solo a coloro che vogliono creare le proprie immagini o a chi è interessato al lavoro generico di programmazione in 3D, ma anche a coloro che vogliono semplicemente vedere alcuni rendering di alta qualità creati da alcuni dei migliori artisti e imparare dal loro codice. Il CD-ROM contiene più di 500 immagini renderizzate. E' una risorsa inestimabile per coloro che stanno imparando POV-Ray e contiene un tutorial interattivo per Windows. Il disco è venduto insieme ad un poster pieghevole ed alla bibliografia. Il CD è compatibile con i formati DOS e Macintosh.
Il CD-ROM è disponibile per 'downloading' e visione sul WWW a *http://www.povray.org/pov-cdrom*. Per maggiori informazioni, visitare l'indirizzo *http://www.povray.org/povcd*.

3. Iniziare Subito

Questo capitolo descrive come installare velocemente POV-Ray e renderizzare le scene d'esempio sul proprio computer. Si assume che si stia usando un computer IBM-PC compatibile con MS-Dos. Per altre piattaforme, si deve fare riferimento alla documentazione specifica contenuta nell'archivio che contiene POV-Ray.

3.1 Installare POV-Ray

Istruzioni specifiche per l'installazione sono fornite insieme al programma eseguibile per la propria piattaforma. In generale, ci sono due modi per installare POV-Ray. (la parola 'directory' è usata per tutto il capitolo. Altri sistemi operativi possono usare altre parole come 'subdirectory, folder, cartella...')
La maniera caotica : creare una directory chiamata **povray** e copiarvi dentro tutti i file. Editare e renderizzare tutte le scene da questa directory. Questo metodo funziona, ma non è raccomandato.
La maniera migliore : creare una directory chiamata **povray** e alcune sottodirectory chiamate **include**, **demo**, **scenes**, **util**. L'archivio autoscompattante che viene usato in alcune versioni del programma creerà automaticamente queste directory. Se vengono create 'a mano' la struttura delle directory dovrebbe essere qualcosa di questo genere :

povray
povray\bin
povray\docsdemo
povray\help
povray\include
povray\ini
povray\pov3demo
povray\povscn
povray\renderer

Copiare il file eseguibile e la documentazione nella directory chiamata **povray**. Copiare i file 'include' (**.inc**) nella sottodirectory **include**. Copiare i file di esempio nella sottodirectory **scenes**, copiare tutti i programmi di utilità e i file ad essi collegati nella sottodirectory **util**. I propri file andranno nella directory **scenes**. Inoltre, andrà modificato il proprio 'path' (percorso di ricerca) aggiungendovi i percorsi **\povray** e **\povray\util** in modo che gli eseguibili possano essere fatti funzionare da qualunque directory.
Si deve fare attenzione al fatto che alcuni sistemi operativi non hanno un equivalente del percorso di ricerca.
Il secondo metodo è un poco più difficile da applicare, ma è preferibile. Ci sono molti file associati a POV-Ray ed è più semplice averci a che fare quando sono separati in sottodirectory diverse.

3.2 Utilizzo di Base

Attenzione : qualora il programma sia stato installato senza servirsi del programma di installazione **install.exe**, gli esempi e le istruzioni qui riportate potrebbero non funzionare. Il processo di installazione configura il file **povray.ini** ed alcuni importanti file batch. Senza questi file correttamente configurati, gli esempi contenuti in questo paragrafo potrebbero non funzionare correttamente.

Lo scopo fondamentale di POV-Ray è di leggere la descrizione di una scena scritta nel linguaggio POV e scrivere un file immagine. I file che descrivono le scene sono file ASCII 'puri' che si creano con un editor di testo. Dozzine di questi file sono inclusi con il programma in modo da illustrarne le varie possibilità.
Il programma viene richiamato scrivendo un comando al prompt di DOS. Il comando è **povray** e deve essere seguito da uno o più 'parametri della linea di comando'. Ogni parametro inizia con un segno + o con un -. I parametri sono separati da uno spazio. I parametri possono essere maiuscoli o minuscoli.
Nota : gli esempi in questo paragrafo assumono che POV-Ray sia stato installato nella directory **c:\povray3**. Il programma di installazione consente di installare POV-Ray in qualunque posizione sul disco e configurerà il programma per qualunque disco e directory siano stati specificati. Si deve solo sostituire agli esempi il proprio drive e la propria directory ovunque sia scritto **c:\povray3.**

Portarsi in quella directory, poi scrivere la seguente linea di comando e premere [INVIO]

```
povray +ishapes +d1
```

Il comando `+i` (i sta per input) dice al programma quale file leggere come input. Se non viene specificata un'estensione al nome del file, si assume l'estensione **.pov**. Quindi, `+ishapes` dice al programma di leggere e renderizzare il file **shapes.pov**.
Il comando `+d1` (d sta per display) dice al programma di mostrare sullo schermo l'immagine. Il comando `-d` disabiliterebbe questa funzione. Il numero '1' dice al programma che tipo di risoluzione utilizzare. Il tipo 1 è la vecchia risoluzione VGA generica a 320 per 200 e solo 256 colori. Funziona su tutti i sistemi VGA. Ci sono altre opzioni attive, oltre a quelle che sono state descritte. Sono raccolte in un file di nome **povray.ini** che è stato creato dal sistema di installazione. POV-Ray cerca automaticamente questo file nella stessa directory dove si trova l'eseguibile. Vedi i paragrafi "Usare i File INI" e "File INI" per maggiori informazioni su **povray.ini** ed altri file **.ini.**
Quando si immette il comando mostrato sopra, si vedono forme geometriche colorate apparire sullo schermo via via che POV-Ray calcola il colore di ogni punto riga dopo riga. Probabilmente il risultato visibile a schermo non sarà soddisfacente. Questo è solo perché si tratta di un'immagine di 'preview'. La vera immagine è a colori a 24-bit., ma non può essere mostrata a schermo utilizzando la modalità VGA con soli 256 colori. Se l'hardware supporta l'interfaccia VESA standard, oppure si ha un driver VESA residente in memoria, si può provare a sostituire `+d1` con `+dg`. Questo darà accesso a tutte le modalità che il proprio hardware video può supportare. Se si ha possibilità di visualizzare le immagini a 15 o 16-bit (in high color) si può provare il comando `+dgh` oppure, avendo un video true color (24 bit) provare `+dgt` per vedere l'immagine al massimo del suo splendore. Vedi il paragrafo "Tipi di Display" per maggiori informazioni sulla preview a schermo.
Quando il programma termina, si sentirà 'beep'. Dopo avere ammirato l'immagine, premere [INVIO]. Comparirà una schermata di statistiche. Se il testo è troppo per entrare nello schermo, si possono premere i tasti cursore per leggere tutte le informazioni. Premere [INVIO] di nuovo per uscire dal programma.
Se non si ha la possibilità di visualizzare le immagini in high color o true color, si dovrà vedere il file immagine per vedere i veri colori. Il file immagine **shapes.tga** si trova nella directory attuale (**c:\povray**). Di regola, POV-Ray crea file in formato TGA. Questo è un formato grafico standard per la memorizzazione di immagini a 24-bit. Sarà necessario un programma di visualizzazione di immagini per vedere il file. Questo genere di programmi è generalmente disponibile alla stessa fonte dove si è trovato POV-Ray, per cui non viene incluso nessun visore col programma.
Se non è possibile visualizzare file TGA, si può aggiungere il comando `+fn` e POV-Ray creerà un file di output in formato PNG (Portable Network Graphics). Se non è disponibile neppure un visore PNG, allora scrivere il seguente comando :

```
t2g shapes
```

e premere [INVIO]. Viene chiamato un programma batch che richiama il programma **tga2gif**. Il programma leggerà il file **shapes.tga**, creerà una tavolozza di 256 colori ottimale e scriverà un file GIF, **shapes.gif**. La maggior parte dei programmi di visualizzazione di immagini supportano il formato GIF.

3.2.1 Renderizzare File in Altre Directory

Normalmente POV-Ray cerca i file di cui ha bisogno solo nella directory corrente. Non cerca file di dati nel path di MS-Dos, ma solo programmi. Nella scena d'esempio appena renderizzata, il file **shapes.pov** era nella directory corrente, per cui non ci sono stati problemi ; quella scena aveva bisogno anche di altri file, ma il file **povray.ini** istruisce POV-Ray sulle altre posizioni in cui cercare i file che gli sono necessari. Se si è permesso al sistema di installazione di aggiornare il file

autoexec.bat, allora è possibile portarsi in qualunque drive, o directory ed eseguire POV-Ray da quella directory. Si potranno anche utilizzare i file batch e i programmi di utilità che sono inclusi nel pacchetto, da qualunque directory. Per futuri riferimenti, chiameremo la strategia "c:\povray3-è-nel-path" strategia 1.
Ci sono alcuni casi in cui si può preferire di non mettere c:\povray3 nel path. C'è un limite di 128 caratteri nella frase PATH= del file **autoexec.bat** e potrebbe quindi non esserci spazio a sufficienza. Si può provare a renderizzare l'esempio **shapes.pov** da un'altra directory. Se non funziona, il computer deve essere riavviato in modo che il nuovo path abbia effetto. Se dopo avere riavviato il computer, non funziona ancora, si dovrà adottare una strategia differente.
Ci sono possibilità che ci siano diverse directory nel path. La maggior parte dei computer hanno **c:\dos** , **c:\windows**,o qualche directory come **c:\utilities** già nel path. Insieme a POV-Ray ci sono diversi piccoli file batch che possono essere copiati in quella directory. Chiameremo questa strategia, "metti-file-batch-in-una-directory-già-nel-path" strategia 2.
Ad un prompt di DOS qualsiasi, scrivere `path` e premere [INVIO]. Verranno mostrate le directory che sono già nel path del sistema operativo. Allora, si dovranno copiare i seguenti file dalla directory **c:\povray3** ad una qualsiasi delle directory già nel path. I file sono : **runpov.bat rerunpov.bat runphelp.bat t2g.bat.**
Una volta copiati questi file, prova il seguente esempio. In questo caso, non fare partire il programma con il comando `povray`. Usa invece `runpov` come segue :

```
cd \povray\pov3demo\showoff
runpov +isunset3 +d1
```

Questo porta alla directory **\povray\pov3demo\showoff** dove si trova il file **sunset3.pov** e fa partire il file **runpov.bat**. Questo file batch è impostato per fare partire POV-Ray anche se l'eseguibile **povray.exe** non si trova nel path. Inoltre passa tutti i comandi contenuti nella riga di comando a **povray.exe**. Questi file batch hanno altri utilizzi, anche se si sta utilizzando la 'strategia 1' come descritta sopra, o la seguente 'strategia 3'. Per maggiori dettagli su questi file batch, vedi "File Batch".

Tutti gli esempi fatti in precedenza assumono che si stesse eseguendo POV-Ray dalla directory nella quale era installato, cioè **c:\povray3**. L'approccio di utilizzare sempre la directory di installazione è in effetti la 'strategia 3' Se stai usando questo metodo, hai bisogno di informare POV-Ray su dove andare a cercare gli altri file. Nel caso di **sunset3.pov** potresti fare così :

```
povray +ic:\povray3\pov3demo\showoff\sunset3 +d1
```

Comunque alcune scene hanno bisogno di più di un file. Per esempio, la directory **\povray3\povscn\level3\drums2** contiene 3 file : **drums.pov**, **drums.inc**, **rednewt.gif** e tutti e tre sono necessari per quella sola scena. In questo caso dovresti usare il comando `+L` (dove L sta per library) per aggiungere nuovi percorsi a quelli che POV-Ray cercherà. In questo caso la scena andrebbe renderizzata con il comando :

```
povray +l\povray3\povscn\level3\drums2 +idrums +d1
```

3.2.2 File INI

Nei rendering appena visti c'erano molte più opzioni attive di quelle che hai specificato con i comandi `+i, +d, +l`. Quando si esegue il programma, POV-Ray cerca automaticamente il file **povray.ini** nella sua stessa directory. Il file **povray.ini** contiene molte opzioni che controllano il funzionamento di POV-Ray. Abbiamo impostato questo file in maniera che sia particolarmente semplice renderizzare la tua prima scena senza problemi. Il file dovrebbe essere messo nella stessa

directory in cui si trova l'eseguibile e verrà automaticamente letto quando verrà fatto girare il programma. Se dovessi spostare **povray.exe** in una directory differente, assicurati di spostare anche **povray.ini**.
Dettagli completi su tutti i comandi e opzioni che possono essere date tramite la linea di comando o il file **povray.ini** sono dati nel capitolo "Opzioni di POV-Ray".
Puoi anche crearti i tuoi file INI con comandi od opzioni simili a **povray.ini.** Se scrivi il nome di un file senza i segni più o meno davanti, POV-Ray lo interpreta come l'indicazione di un file INI. Prova questo...

```
povray res120 +ishapes +d1
```

Questo fa sì che POV-Ray cerchi un file chiamato **res120.ini** che abbiamo fornito. Questo file regola la risoluzione a 120 punti per 90 per dare una preview molto veloce. Sono disponibili i seguenti file :

res120.ini regola la risoluzione a 120 per 90
res320.ini regola la risoluzione a 320 per 200
res640.ini regola la risoluzione a 640 per 480
res800.ini regola la risoluzione a 800 per 600
res1k.ini regola la risoluzione a 1024 per 768
low.ini regola la risoluzione a 120 per 90 a bassa qualità
slow.ini attiva radiosity ed antialiasing. Molto lento ma bello
tgafli.ini crea un'animazione da immagini TGA
tgaflc.ini idem
pngfli.ini crea un'animazione da immagini PNG
pngflc.ini idem
zipfli.ini crea un'animazione da immagini compresse con pkzip
zipflc.ini idem

Puoi creare i tuoi file INI personalizzati che contengano qualunque comando citato nella guida.

3.2.3 Alternative a POVRAY.INI

Il file **povray.ini** dovrebbe contenere tutte le tue opzioni preferite, quelle che usi ogni volta. Per questo, ti dovresti sentire libero di editarlo con le nuove opzioni che possono soddisfare le tue necessità. Comunque, deve trovarsi nella stessa directory di **povray.exe** o non verrà trovato. Il path di MS-Dos non viene analizzato, così come l'aggiunta di un comando `+l` non aiuta, perché **povray.ini** viene letto da POV-Ray prima di qualunque altro comando. Se la tua copia di **povray.exe** si trova su un CD-ROM, allora non puoi editare **povray.ini** sul CD. C'è un'alternativa. Puoi utilizzare una variabile di ambiente per specificare un valore alternativo. Nel tuo file **autoexec.bat** aggiungi una linea simile a questa :

```
set POVINI=d:\direct\file.ini
```

che regola la variabile d'ambiente POVINI verso un qualunque drive, directory e file INI che tu hai scelto. Se specifichi un qualsiasi valore per la variabile d'ambiente POVINI, allora POV-Ray non legge **povray.ini.** Questo accade anche se il file INI che hai specificato non esiste. Nota che si deve specificare un percorso completo e il nome del file. Questo non è un puntatore ad una directory contenente **povray.ini**, è un puntatore al file in sé.
Nota anche che la variabile di ambiente POVRAYOPT delle precedenti versioni di POV-Ray non è più supportata.

3.2.4 File Batch

Abbiamo già descritto come il file **runpov.bat** può essere utilizzato come un'alternativa per fare funzionare POV-Ray direttamente. **runpov.bat** ha anche un altro utilizzo. Usa il comando `+gi` per creare un file chiamato **rerun.ini**. Questo rende molto semplice renderizzare lo stesso file con gli stessi parametri. Quando crei le tue scene, probabilmente farai dozzine di rendering di prova. Questa è una funzione molto comoda. Ecco come funziona...supponi di renderizzare una scena in questo modo :

```
runpov +iscena +d1 res120
```

Questa linea di comando renderizza **scena.pov** ad una risoluzione di 120 per 90. Nota che questa scena (cioè il file **scena.pov**) non è incluso nel programma. E' ipotetico ; dopo aver guardato l'immagine, hai notato un errore che hai corretto con l'editor di testo. Per renderizzare nuovamente la scena scrivi :

```
rerunpov
```

ed è tutto. Verrà renderizzata la stessa scena di prima con le stesse impostazioni. Supponi di volere un maggior dettaglio alla prossima prova. Puoi aggiungere altri comandi o file INI, come ad esempio

```
rerunpov res320
```

darà un rendering a risoluzione maggiore. Successivi utilizzi di **rerunpov** daranno immagini a 320 punti per 200 fino a che non specificherai diversamente. Un altro esempio : il comando `+A` inserisce l'antialiasing. Scrivere "`rerunpov +A`" fa partire lo stesso rendering con l'antialiasing. Tutte le volte successive saranno con antialiasing fino a che non specificherai "`rerunpov -A`" per disattivarlo. Nota che un successivo `runpov` riparte con le opzioni contenute nel tuo **povray.ini** e sovrascrive il vecchio **rerun.ini**.
Sono inclusi altri due file batch. **runphelp.bat** è utilizzato solo come alternativa per potere usare **povhelp** da un'altra directory. Se hai usato la 'strategia 2' allora usa **runphelp.bat** invece di **povhelp.exe**. Questo file batch non serve ad altro.
Infine, **t2g.bat** fa partire il programma **tga2gif.exe** per convertire immagini TGA ad immagini GIF. Potresti usare direttamente **tga2gif.exe** ma i suoi parametri predefiniti generalmente non danno i risultati migliori. Se invece utilizzi **tga2gif** vengono aggiunti alcuni comandi che lo migliorano. Per una lista completa dei comandi e delle opzioni disponibili per il programma, scrivi **tga2gif** senza parametri.

3.2.5 Tipi di Display

Abbiamo già visto come attivare un'anteprima dell'immagine utilizzando il comando +d1. Ecco altri dettagli sulle possibili variazioni sul comando `+d`. Usa `-d` per disattivare l'uscita a schermo. Se usi il comando `-d` probabilmente vorrai utilizzare `+v` per attivare i messaggi (modalità "verbose") in modo da tenere sotto controllo l'andamento del rendering.
Il numero "1" dopo il comando `+d` indica il tipo di hardware video da utilizzare. Usando il comando `+d` , oppure `+d0`, POV-Ray cercherà di identificare da solo il tuo hardware video. Usa `+d?` per avere un messaggio indicante il tipo di hardware che POV-Ray ha trovato. Puoi anche indicare esplicitamente a POV-Ray quale tipo di hardware video utilizzare. Ecco una lista di tutti i tipi supportati :

comando	scheda video
+D0	identificazione automatica

+D1	Standard VGA 320x200
+D2	Standard VGA 360 x 480
+D3	Tseng Labs 3000 SVGA 640x480
+D4	Tseng Labs 4000 SVGA
+D5	AT&T VDC600 SVGA 640x400
+D6	Oak Technologies SVGA 640x480
+D7	Video 7 SVGA 640x480
+D8	Video 7 Vega (Cirrus) VGA 360x480
+D9	Paradise SVGA 640x480
+DA	Ahead Systems Ver. A SVGA 640x480
+DB	Ahead Systems Ver. B SVGA 640x480
+DC	Chips & Technologies SVGA 640x480
+DD	ATI SGVA 640x480
+DE	Everex SVGA 640x480
+DF	Trident SVGA 640x480
+DG	VESA Standard SVGA
+DH	ATI XL scheda video
+DI	Diamond Computer Systems SpeedSTAR 24X

Il genere di hardware video più diffuso è la scheda standard VESA, che utilizza il comando `+dg`. VESA è un'interfaccia software standard che funziona su di un'ampia gamma di schede video. Le schede che non hanno supporto diretto alle specifiche VESA, generalmente hanno un driver video che può essere caricato in memoria per ottenere supporto VESA. Il programma **UniVBE** è un driver VESA universale di alta qualità che dovrebbe funzionare per tutti. Puoi trovarlo a *http://www.povray.org* o altri siti Internet di POV-Ray.
Le opzioni elencate sopra sono state provate nelle prime versioni di POV-Ray ma ci sono stati molti cambiamenti nel programma e non possiamo garantire che tutte funzionino. Se puoi usare lo standard VESA, fallo. E' stato provato approfonditamente e fornirà la maggiore flessibilità.
Dopo il comando `+d`, si può specificare un terzo carattere che specifica il tipo di tavolozza da utilizzare.

`+d?3` Utilizza una tavolozza di tipo 332 (predefinita, la migliore per i sistemi video VGA). E' una tavolozza fissa a 256 colori con 3 bit per i dati del rosso, 3 per il verde e 2 per il blu.
Usa una tavolozza HSV per schede VGA. E' una tavolozza fissa a 256 colori in cui i colori sono individuati per tonalità (hue) saturazione (saturation) e intensità invece che per le componenti di rosso, verde, blu.
`+d?g` Usa una tavolozza a toni di grigio per video VGA.
`+d?h` Usa l'opzione hi-color. Mostra a video più di 32000 colori con dithering. Viene supportato da schede video VESA, SpeedSTAR 24X, ATI XL HiColor e con processore video Tseng 4000 con opzione per high color a 15 o 16 bit.
`+d?t` Per schede video true color. Usa colore a 24 bit. E' supportata dalla Diamond Speedstar 24X e da schede con supporto VESA a 24 bit.

Ecco alcuni esempi :
`+d0h` Riconosci automaticamente la scheda video e mostra a schermo l'immagine mentre viene renderizzata. Usa 15 bit per pixel (mostra l'immagine con 32000 colori differenti).
`+d4` Mostra l'immagine su una scheda video con chip grafico TSENG 4000 usando la tavolozza

332.
`+d4h` Mostra l'immagine su una scheda video con chip grafico TSENG 4000 usando l'opzione hi-color.
`+dg0` Mostra l'immagine su una scheda video VGA usando la tavolozza HSV.
`+dg3` Mostra l'immagine su una scheda video VGA usando la tavolozza 332.
`+dgh` Mostra l'immagine su una scheda video VGA usando l'opzione hi-color.
`+dgt` Mostra l'immagine su una scheda video VGA usando l'opzione true color.
Nota che il BIOS VESA deve supportare queste opzioni affinché tu le possa usare. Alcune schede possono supportare hi-color o true color a livello hardware, ma non tramite il loro BIOS VESA.

4. Tutorial

Questo capitolo spiega passo dopo passo come utilizzare il linguaggio di cui POV-Ray fa uso per creare le proprie scene. Verrà spiegato in dettaglio l'utilizzo di quasi tutte le funzioni del linguaggio di POV-Ray. Impareremo funzioni elementari come posizionare il punto di osservazione e le sorgenti luminose, impareremo anche come creare una grande varietà di oggetti ed assegnare loro texture differenti. Le funzioni più sofisticate come radiosity (riflessione interdiffusa), aloni (oggetti particellari) ed effetti atmosferici verranno spiegate in dettaglio. I paragrafi seguenti spiegano le funzioni all'incirca nello stesso ordine in cui compaiono nella guida di riferimento (vedi capitolo 7).

4.1 La Nostra Prima Immagine

Creeremo un file scena per un'immagine semplice. Dato che i programmi di raytracing si basano sulle sfere, partiremo da lì.

4.1.1 Capire il sistema di coordinate di POV-Ray

Per prima cosa, dobbiamo dire a POV-Ray dove si trova il nostro punto di osservazione e dove stiamo guardando (il punto di osservazione è indicato dall'istruzione `camera`, che significa 'macchina fotografica'). Per fare questo utilizziamo un sistema di coordinate a 3 dimensioni. Il normale sistema di coordinate per POV-Ray ha l'asse y rivolto verso l'alto, l'asse x rivolto a destra e l'asse z rivolto verso l'interno dello schermo. Questo tipo di sistema di coordinate viene detto **sinistrorso :** utilizzando per un momento le dita della nostra mano sinistra possiamo capire perché.

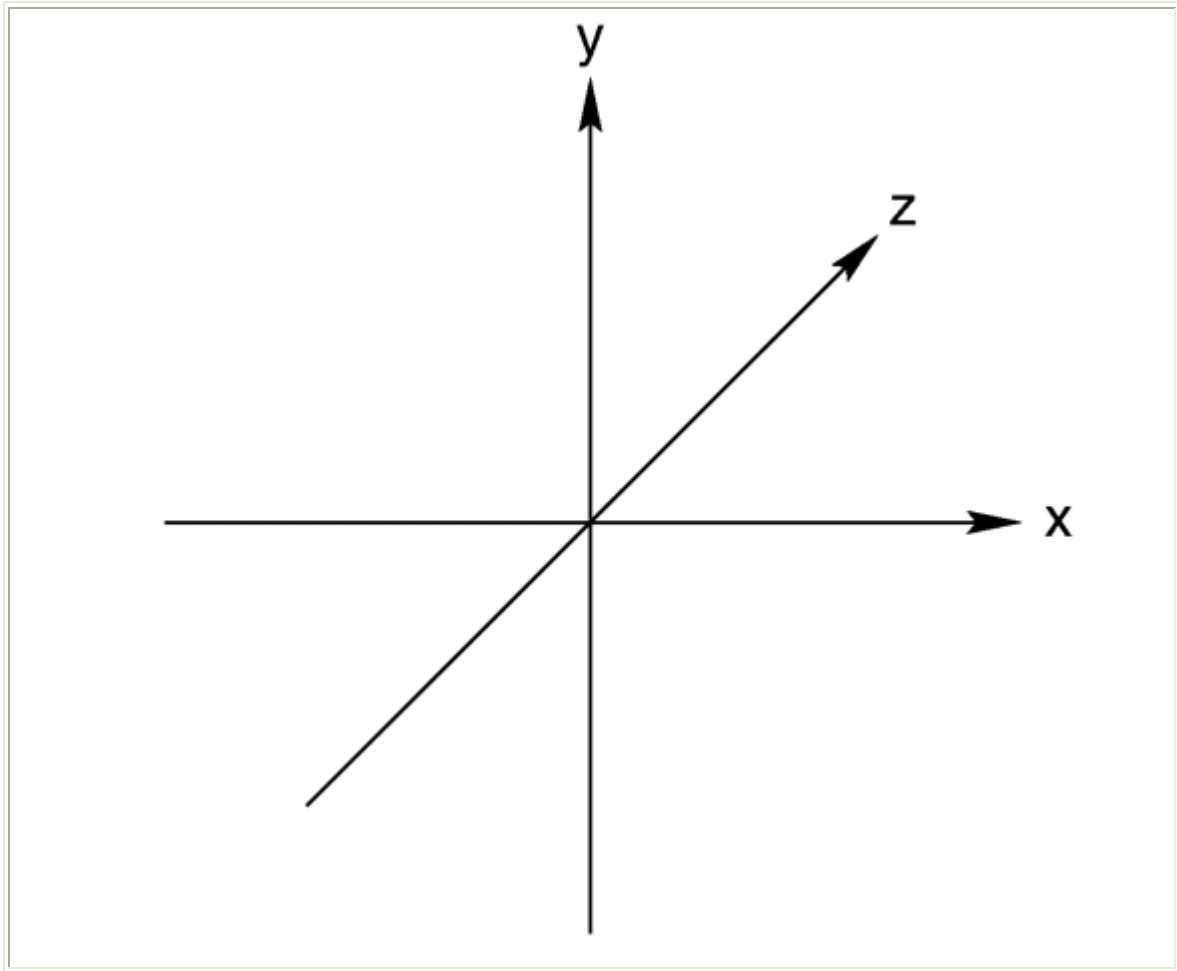



Fig. 1-Il sistema di coordinate di POV-Ray

Puntiamo il pollice nella direzione dell'asse delle x positive, l'indice nella direzione delle y positive ed il medio nella direzione delle z positive. Solo la mano sinistra lo permette, se avessimo utilizzato la destra non avremmo potuto rivolgere il medio nella direzione giusta. La mano sinistra può anche essere utilizzata per determinare la direzione delle rotazioni. Per fare questo dobbiamo eseguire il famoso esercizio di 'Aerobica per Grafica Computerizzata'.

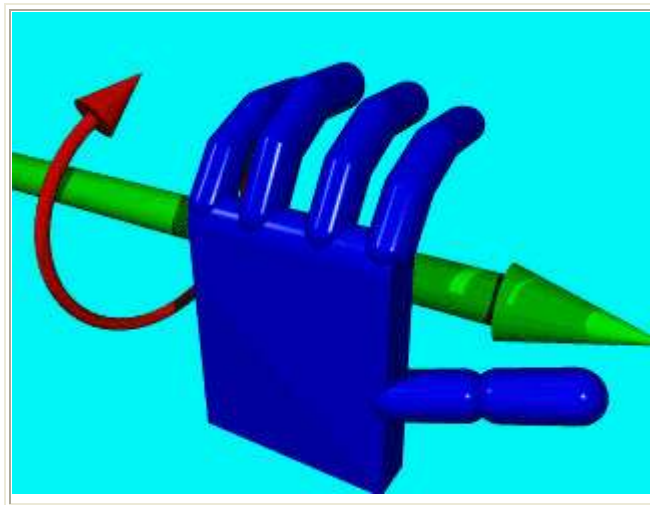



Fig.2

Solleviamo la nostra mano sinistra e puntiamo il pollice nella direzione positiva dell'asse di rotazione. Le altre dita si piegheranno nella direzione corrispondente al verso positivo della

rotazione, dal dorso verso le punte. Se puntiamo il pollice nell'altro verso dell'asse di rotazione, otterremo la direzione di rotazione negativa.
Ad esempio, la mano sinistra si avvolge attorno all'asse delle x. Il pollice punta nella direzione positiva dell'asse delle x e le dita si ripiegano nella direzione della rotazione. Se volessimo utilizzare un sistema di assi di riferimento destrorso, come avviene in alcuni sistemi CAD e modellatori, il vettore che indica la direzione di destra nella definizione della `camera` dovrebbe venire modificato. Vedi la descrizione dettagliata nel paragrafo "Chiralità". In un sistema destrorso viene utilizzata la mano destra per svolgere gli esercizi di 'aerobica'. C'è in atto qualche controversia nel modo in cui POV-Ray simula un sistema di coordinate destrorso. Nel dubbio, ci teniamo il sistema di coordinate sinistrorso, sul quale non ci sono problemi.

4.1.2 Aggiungere File 'include' Standard

Utilizzando il nostro editor di testo preferito, creiamo un file che chiameremo **demo.pov**. Poi vi scriviamo il seguente testo. L'input è sensibile alla differenza tra maiuscole e minuscole, quindi dovremo assicurarci che tutte le maiuscole e minuscole siano corrette.

```
#include "colors.inc"
#include "shapes.inc"
#include "finish.inc"
#include "glass.inc"
#include "metals.inc"
#include "stones.inc"
#include "woods.inc"
```

I file 'include', che da qui chiameremo **.inc** contengono elementi predefiniti per le scene. L'istruzione contenuta nella prima riga fa leggere a POV-Ray le definizioni per vari colori utili. La seconda istruzione (seconda riga) fa leggere le definizioni per solidi e forme geometriche. Le successive leggono finiture e informazioni per ottenere materiali simili a vetri, metalli, pietre, legni. Può essere una buona idea leggerli per vedere almeno poche delle texture e forme geometriche disponibili.
Si dovrebbero includere solo i file di cui abbiamo veramente bisogno nelle nostre scene. Alcuni dei file che sono inclusi con POV-Ray sono molto grandi ed è meglio risparmiare tempo di calcolo e memoria se non ne abbiamo bisogno. Negli esempi seguenti, utilizzeremo solo i file **colors.inc**, **finish.inc**, **stones.inc** e quindi è meglio rimuovere le linee corrispondenti agli altri dal nostro file scena. Possiamo avere quanti file **.inc** desideriamo nel nostro file scena. Gli stessi file **.inc** possono contenere al loro interno frasi `#include`, ma abbiamo la limitazione di effettuare questi "include annidati" solo fino al decimo livello. I file specificati in una frase `#include` verranno ricercati prima nella directory attuale e, se non trovati, nelle directory specificate da un comando `+l` o da un'opzione `Library_Path`. Quindi ci faciliterebbe le cose tenere tutti i file **.inc** come **colors.inc**, **textures.inc** ecc. in una directory **include** e fornire un comando `+l` che specifichi dove è questa directory.

4.1.3 Aggiungere la Macchina Fotografica

La dichiarazione della 'camera' descrive dove e come la macchina fotografica vede la scena. Fornisce coordinate x, y, z per indicare la posizione della 'camera' e il punto verso cui è rivolta. Descriviamo le coordinate spaziali di questi punti usando un vettore a tre componenti. Un vettore viene specificato utilizzando tre valori numerici separati da virgole e racchiusi in parentesi angolate. Aggiungiamo pertanto la seguente frase `camera` alla scena :

```
camera {
location <0, 2, -3>
```

```
look_at <0, 1, 2>
}
```

In breve, `location <0, 2, -3>` posiziona la macchina fotografica due unità in alto e tre unità indietro rispetto al centro dell' 'universo', che si trova a `<0, 0, 0>`. Per convenzione, la direzione +z è rivolta all'interno dello schermo e la direzione -z punta 'fuori' dallo schermo. L'istruzione `look_at <0, 1, 2>` ruota la macchina fotografica in modo che punti verso il punto di coordinate `<0, 1, 2>`. Un punto 5 unità davanti ed 1 unità sotto la macchina fotografica. Il punto `look_at` dovrebbe essere il centro di attenzione della nostra immagine.

4.1.4 Descrivere un Oggetto

Ora che abbiamo piazzato una macchina fotografica per osservare la scena, mettiamo una sfera gialla nella scena. Aggiungiamo il seguente testo al nostro file :

```
sphere {<0, 1, 2>, 2
texture {
pigment { color Yellow }
}
}
```

Il primo vettore specifica il centro della sfera. In questo esempio, la coordinata x è zero, così che la sfera risulti centrata a sinistra ed a destra. Si trova anche a y=1, ovvero un'unità in alto rispetto all'origine. La coordinata z è 2, che significa 5 unità davanti alla macchina fotografica (che si trova a z=-3). Dopo il vettore che indica il centro della sfera, si ha una virgola seguita dal raggio, che in questo caso è 2. Dato che il raggio è la metà del diametro di una sfera, la nostra sfera ha un diametro di quattro unità.

4.1.5 Aggiungere una Texture ad un Oggetto

Dopo avere definito posizione e dimensioni della sfera, dobbiamo descrivere come appare la sua superficie. Ciò è descritto dal blocco di istruzioni `texture`. I blocchi `texture` descrivono il colore, la 'ruvidezza' e la finitura di un oggetto. In questo esempio specificheremo solo il colore. Questo è il minimo che dobbiamo fare. Tutte le altre opzioni eccetto il colore utilizzeranno valori predefiniti. Il colore che definiamo è il modo in cui vogliamo che un oggetto appaia se è illuminato. Se stessimo dipingendo un quadro che rappresenta una sfera, utilizzeremmo toni scuri di un colore per indicare la parte in ombra e toni chiari dello stesso colore per indicare la parte illuminata. Comunque, il raytracing si preoccupa di questo. Noi scegliamo il colore base per l'oggetto e POV-Ray lo schiarisce o scurisce in dipendenza dall'illuminazione della scena. Dato che stiamo definendo il colore base dell'oggetto, più che il suo aspetto, il parametro si chiama `pigment`. Molti tipi di colore sono disponibili per essere utilizzati in una frase `pigment`. La parola `color` specifica che l'intero oggetto è in un unico colore invece che essere coperto con un qualche motivo decorativo in più colori. Possiamo utilizzare gli identificatori di colore precedentemente definiti nel file **colors.inc**. Se non c'è un colore standard che si adatti alle nostre esigenze, posiamo definire i nostri colori personalizzati, utilizzando la parola `color` seguita dalle parole `red`, `green`, `blue` che specificano l'ammontare delle componenti di rosso, verde e blu da mescolare. Ad esempio, una bella tonalità di rosa può essere specificata usando

```
color red 1.0 green 0.8 blue 0.8
```

I valori dopo ogni 'keyword' (parola chiave, in questo caso `red, green, blue`) dovrebbero essere compresi tra 0 ed 1. Se qualcuna delle tre componenti non venisse specificata, il suo valore verrebbe interpretato come zero. Si può usare anche una notazione abbreviata. La seguente frase

produce lo stesso rosa :

```
color rgb <1.0, 0.8, 0.8>
```

I colori sono spiegati in maggiore dettaglio nella sezione "Specificare i Colori".

4.1.6 Definire una Sorgente Luminosa

Abbiamo bisogno di un'altra cosa per la nostra scena. Abbiamo bisogno di una sorgente luminosa. Fino a che non ne creiamo una, non c'è luce in questo mondo virtuale. Quindi, aggiungiamo la linea :

```
light_source { <2, 4, -3> color White}
```

alla nostra scena in modo da ottenere il nostro primo file scena per POV-Ray come segue :

```
#include "colors.inc"
background { color Cyan}
camera {
location <0, 2, -3>
look_at <0, 1, 2>
}
sphere {
<0, 1, 2>, 2
texture {
pigment { color Yellow }
}
}
light_source { <2, 4, -3> color White}
```

Il vettore nella frase `light_source` specifica la posizione della luce due unità alla nostra destra, quattro sopra l'origine e tre indietro rispetto all'origine. La sorgente luminosa è invisibile : emette solamente luce, per cui non è necessario assegnarle una texture.

Ecco fatto ! Chiudiamo il file e renderizziamo una piccola immagine utilizzando il comando :

```
povray +w160 +h120 +p +x +d0 -v -idemo.pov
```

Se il nostro computer non usa le istruzioni a riga di comando, si devono leggere le istruzioni per la specifica piattaforma, per fornire il comando corretto per renderizzare la scena. Possiamo anche regolare qualunque altro comando che si possa ritenere utile. L'immagine è scritta in un file immagine di nome **demo.tga** (o qualche altro suffisso differente da **.tga** se il nostro computer utilizza un formato differente).

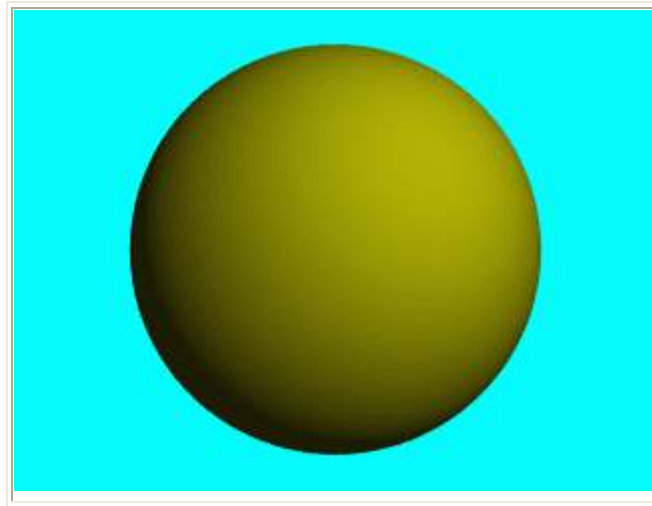



Fig. 3-Prima Immagine

Quest'immagine non è esattamente un'opera d'arte, ma bisognerà partire con gli elementi basilari prima di arrivare a scene e funzioni molto più affascinanti.

4.2 Usare la Macchina Fotografica

4.2.1 Usare la Messa a Fuoco

Costruiamo una semplice scena per mostrare l'impiego della messa a fuoco. Per questo esempio useremo una sfera rosa, un parallelepipedo verde ed un cilindro blu, con la sfera davanti, il parallelepipedo in mezzo ed il cilindro dietro. Un pavimento a scacchiera per evidenziare la prospettiva ed un paio di sorgenti luminose completeranno la scena.
Creiamo un nuovo file chiamato **focaldem.pov** e vi inseriamo il seguente testo.

```
#include "colors.inc"
#include "shapes.inc"
#include "textures.inc"
#version 3.0
#global_settings {
assumed_gamma 2.2
max_trace_level 5
}
sphere { <1, 0, -6>, 0.5
finish {
ambient 0.1
diffuse 0.6
}
pigment { NeonPink }
}
box { <-1, -1, -1>, < 1, 1, 1>
rotate <0, -20, 0>
finish {
ambient 0.1
diffuse 0.6
}
pigment { Green }
}
cylinder { <-6, 6, 30>, <-6, -1, 30>, 3
```

```
finish {
ambient 0.1
diffuse 0.6
}
pigment {NeonBlue}
}
plane { y, -1.0
pigment {
checker color Gray65 color Gray30
}
}
light_source { <5, 30, -30> color White }
light_source { <-5, 30, -30> color White }
```

Ora possiamo procedere a piazzare la nostra macchina fotografica in una posizione appropriata. Mettendola dietro i nostri tre oggetti darà un buon risultato. Aggiustare il punto focale, lo muoverà in un qualunque punto della scena. Aggiungiamo solo le righe seguenti al file :

```
camera {
location <0.0, 1.0, -10.0>
look_at <0.0, 1.0, 0.0>
// focal_point <-6, 1, 30> // cilindro blu a fuoco
// focal_point < 0, 1, 0> // parallelepipedo verde a fuoco
focal_point < 1, 1, -6> // sfera rosa a fuoco

aperture 0.4 // un buon compromesso
// aperture 0.05 // quasi tutto è a fuoco
// aperture 1.5 // molto sfuocato

// blur_samples 4 // pochi campioni, più veloce
blur_samples 20 // più campioni, immagine di alta qualità
}
```

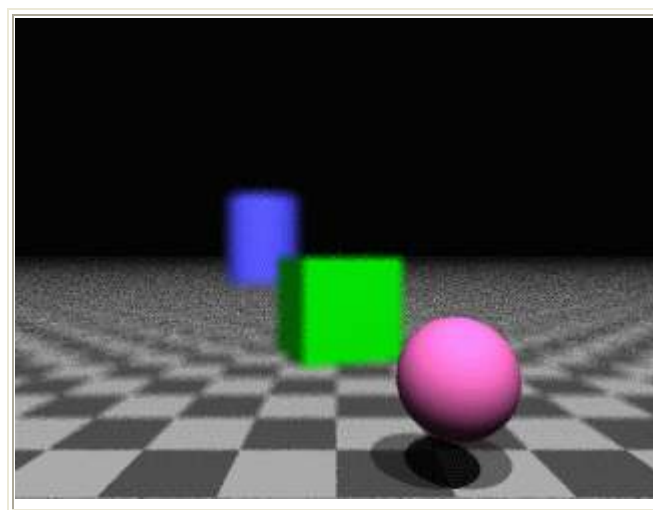



Fig.4-Messa a fuoco

Il punto focale è semplicemente il punto dove il fuoco della macchina fotografica è massimo e l'immagine risulta più nitida. Posizioniamo questo punto nella nostra scena ed assegnamo un valore all'apertura per determinare quanto vicino o quanto lontano dal punto di massimo fuoco vogliamo che appaia la sfocatura. Aumentare l'apertura ha l'effetto di rendere minore l'area a fuoco, mentre

diminuirla rende maggiore l'area dell'immagine a fuoco. Questo è il modo in cui controlliamo come la sfocatura comincia ad avvenire attorno al punto focale. Il valore `blur_samples` determina quanti raggi sono usati per campionare ogni singolo punto dell'immagine (pixel). In maniera semplificata, quanti più raggi si usano, tanto migliore è la qualità dell'immagine risultante, ma conseguentemente aumenta anche il tempo necessario per il rendering. Ogni scena è diversa dalle altre e così dovremo sperimentare. Questo tutorial ha due esempi da 4 e 20 campioni, ma ne possiamo utilizzare di più per immagini ad alta risoluzione. Non si dovrebbero usare più campioni di quanti siano necessari per ottenere la qualità desiderata, dato che maggiore è il numero dei campioni, maggiore è il tempo di rendering. I valori di `confidence` e `variance` sono trattati nella sezione "Messa a Fuoco" della guida al linguaggio.
Conviene sperimentare con le regolazioni dei valori `focal_point`, `aperture`, `blur_sample`. La scena ha linee con altri valori che possiamo provare commentando la linea di default con un doppio slash (//) e de-commentando la linea che vogliamo provare. Facciamo solo un cambiamento per volta, per vedere gli effetti sulla scena.
Due cose finali da sapere quando si usa la messa a fuoco : primo, non abbiamo bisogno di specificare l'anti-aliasing (con il comando `+a`) dato che il metodo di messa fuoco si occupa automaticamente dell'antialiasing ; secondo, la messa a fuoco può venire usata solo per 'macchine fotografiche' di tipo prospettico.

4.3 Oggetti Semplici

Fino ad ora abbiamo visto solo l'oggetto 'sfera'. Oltre alla sfera, ci sono molti altri tipi di oggetti che possono essere renderizzati da POV-Ray. I paragrafi seguenti descriveranno come utilizzare alcuni degli oggetti più semplici come sostitutivi della sfera usata prima.

4.3.1 Parallelepipedo

Il parallelepipedo è uno degli oggetti più usati. Proviamo questo esempio al posto della sfera.

```
box {
<-1, 0, -1>, // angolo inferiore sinistro (più vicino)
< 1, 0.5, 3> // angolo superiore destro (più lontano)
texture {
T_Stone25 // predefinita da stones.inc
scale 4 // ingrandita nello stesso modo in tutte le direzioni
}
rotate y*20 // Equivalente a "rotate <0,20,0>"
}
```

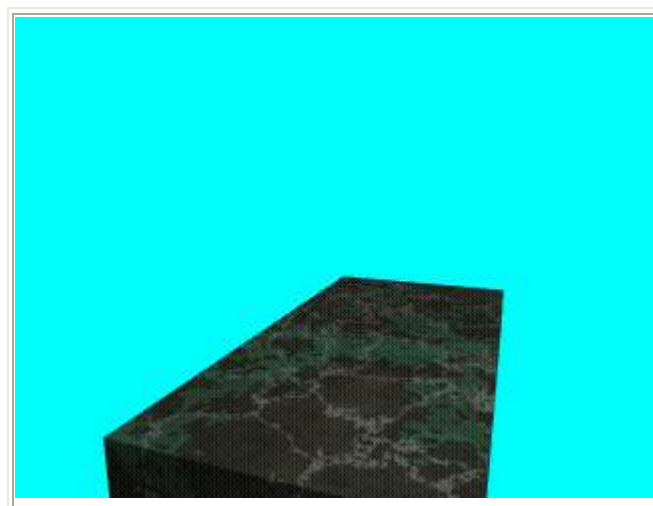



Fig. 5-Parallelepipedo

In questo esempio possiamo vedere che un parallelepipedo (una "scatola", cioè box) viene definito specificando le coordinate tridimensionali di due suoi vertici opposti. Il primo vettore deve essere formato dalle coordinate minime x, y, z e il secondo dalle coordinate massime. Si vede come l'oggetto `box` possa essere definito solo parallelamente agli assi di riferimento. In seguito, lo possiamo ruotare di un qualsiasi angolo. Nota che possiamo eseguire semplici operazioni matematiche sui vettori e sui valori numerici. Per esempio, nel parametro `rotate` abbiamo moltiplicato l'identificatore y per 20. Questo è lo stesso che scrivere `<0,1,0>*20` oppure `<0,20,0>`.

4.3.2 Cono

Ecco un esempio su come usare un cono :

```
cone {
<0, 1, 0>, 0.3 // centro e raggio di un'estremità
<1, 2, 3>, 1.0 // centro e raggio dell'altra estremità
texture { T_Stone25 scale 4 }
}
```

Fig.6-Cono

Un cono (o tronco di cono) viene definito mediante il centro ed il raggio delle sue due estremità. In questo esempio, un'estremità si trova a <0,1,0> e ha raggio 0.3, mentre l'altra si trova a <1,2,3> con raggio 1. Se vogliamo che il cono abbia una punta, dobbiamo porre uguale a zero uno dei due raggi. Le facce piane del cono sono parallele l'una all'altra e perpendicolari all'asse del cono. Se vogliamo un cono senza queste facce, dobbiamo aggiungere la parola `open` dopo il secondo raggio, in questo modo :

```
cone {
<0, 1, 0>, 0.3 // centro e raggio di un'estremità
<1, 2, 3>, 1.0 // centro e raggio dell'altra estremità
open // elimina le facce piane
texture { T_Stone25 scale 4 }
}
```

Fig.7-Cono senza le facce piane (`open`)

4.3.3 Cilindro

Possiamo anche definire un cilindro come questo :

```
cylinder {
<0, 1, 0>, // centro di un'estremità
<1, 2, 3>, // centro dell'altra estremità
0.5 // raggio
open // elimina le facce piane
texture { T_Stone25 scale 4 }
}
```

Fig. 8-Cilindro

4.3.4 Piano

Proviamo uno degli standard della computer grafica : il piano a scacchi. Aggiungiamo il seguente oggetto alla prima versione del file **demo.pov**, quello con la sfera.

```
plane{ <0,1,0> ,-1
pigment{
```

```
checker color Red color Blue
}
}
```

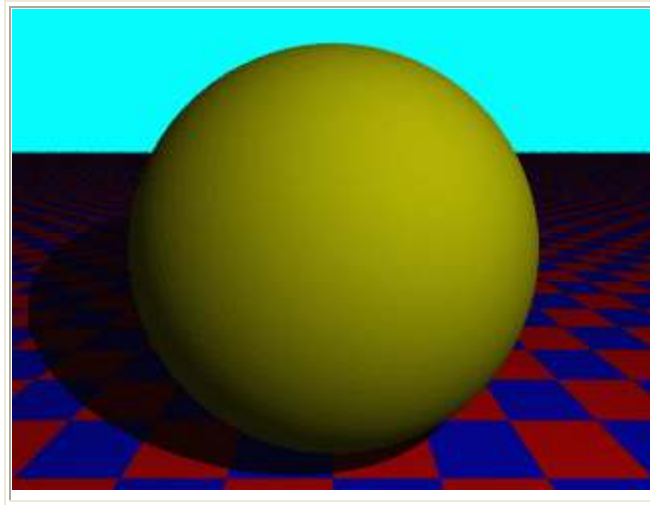



Fig. 9-Piano

L'oggetto qui definito è un piano infinito. Il vettore <0,1,0> è la normale alla superficie del piano (cioè, se fossimo in piedi sulla superficie, vedremmo questo vettore dirigersi verso l'alto). Il numero che segue è la distanza di cui il piano è spostato, lungo la normale, dall'origine. In questo caso, il piano è posto ad y=-1 in modo che la sfera, centrata a y=1, di raggio 2 sia appoggiata su di esso.
Notiamo che anche se qui non c'è una frase `texture` in effetti c'è una texture implicita. Potremmo accorgerci che scrivere frasi che possono essere annidate, come `texture{ pigment}` può diventare stancante, così POV-Ray ci permette di tralasciare la frase `texture` in molti casi. In generale, abbiamo bisogno dei blocchi `texture` solo quando dobbiamo inserire un identificatore di texture, come il precedente `T_Stone25` oppure quando dobbiamo costruire texture a strati (vedi § 4.8.2.4 e § 7.6.6).
Questo pigmento utilizza il motivo "checker" (scacchiera) e specifica di disporre a scacchiera i due colori rosso e blu.
Dato che i vettori <1,0,0>, <0,1,0>,<0,0,1> sono usati frequentemente, POV-Ray ha 'incorporati' tre identificatori per questi vettori, chiamati x, y, z rispettivamente, che possono essere utilizzati come abbreviazione. Allora, il piano potrebbe essere definito come :

```
plane {y, -1
pigment{...}
}
```

Nota che non usiamo le parentesi angolate <> attorno agli identificatori dei vettori. Guardando il pavimento, notiamo che la sfera forma un'ombra sul pavimento. Le ombre sono calcolate molto accuratamente dal programma, che traccia ombre molto precise, nette. Nel mondo reale, sono molto più diffuse ombre 'morbide' e penombra. Più tardi impareremo come utilizzare sorgenti luminose 'estese' per ammorbidire le ombre.

4.3.5 Oggetti Standard

Il file **shapes.inc** contiene alcuni oggetti predefiniti che sono circa delle dimensioni di una sfera di raggio 1. Possiamo utilizzarli in questo modo :

```
#include "shapes.inc"
object{
```

```
UnitBox
texture { T_Stone25 scale 4}
scale 0.75
rotate <-20,25,0>
translate y
}
```

4.4 Oggetti Avanzati

Dopo avere fatto un po' di esperienza con gli oggetti più semplici disponibili in POV-Ray, è ora di proseguire con oggetti più avanzati (ed emozionanti). Dovremmo fare attenzione al fatto che gli oggetti descritti più avanti non sono banali da comprendere. Non c'è bisogno di preoccuparsi se non sappiamo come usarli o come funzionano. Per ora, proviamo gli esempi e giochiamo con le opzioni che sono spiegate più approfonditamente nel capitolo dedicato al linguaggio. Non c'è niente di meglio che imparare facendo.

4.4.11 Toro

Un toro può essere pensato come una ciambella o un tubo richiuso. E' un oggetto molto utile in molti tipi di costruzioni CSG, così POV-Ray ha adottato questa superficie polinomiale di 4° ordine come una primitiva. La sintassi per il toro è talmente semplice da renderlo un oggetto molto comodo con cui lavorare, una volta imparato cosa significano i due valori numerici. Invece di fare una lezione sull'argomento, facciamone uno e sperimentiamoci un po' sopra.
Creiamo un file di nome **tordemo.pov** e lo editiamo come segue :

```
#include "colors.inc"
camera {
location <0, .1, -25>
look_at 0
angle 30
}

background { color Gray50 } // per vederlo meglio
light_source{ <300, 300, -1000> White }
torus { 4, 1 // raggi maggiore e minore
rotate -90*x // per vederlo da sopra
pigment { Green }
}
```

Tracciamo la scena e vediamo cosa succede.

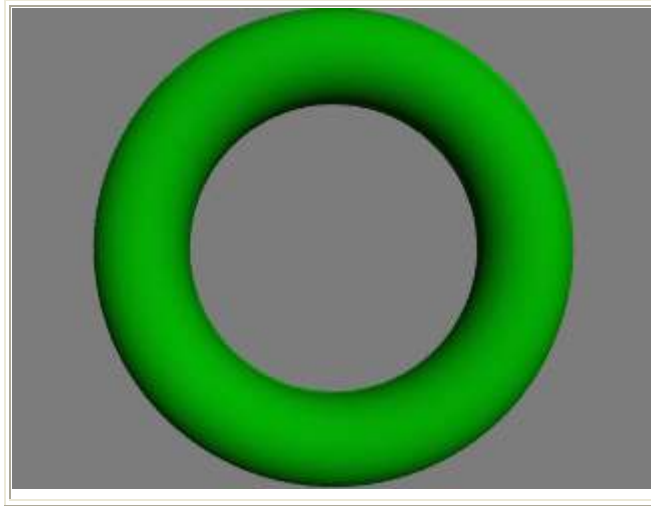



Fig. 52-Toro

Bene, è una ciambella. Proviamo a cambiare i valori del raggio maggiore e minore e vediamo cosa succede. Li modifichiamo come segue :

```
torus { 5, .25 // raggi maggiore e minore
```

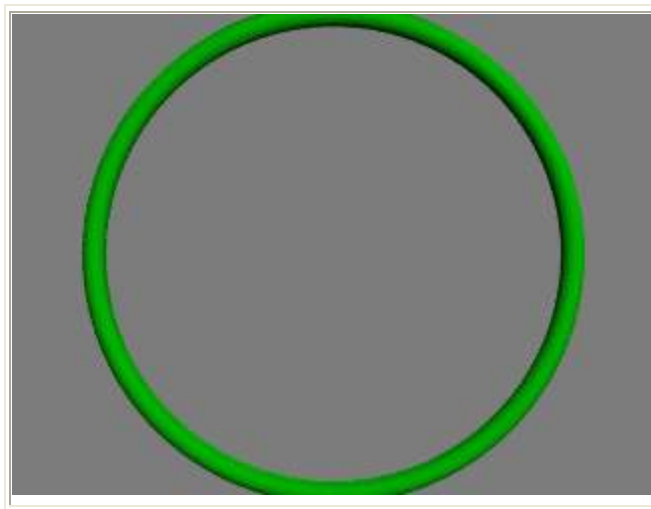



Fig. 53- Toro più sottile

Questo sembra di più un hula-hoop. Proviamo questo :

```
torus { 3.5, 2.5 //raggi maggiore e minore
```

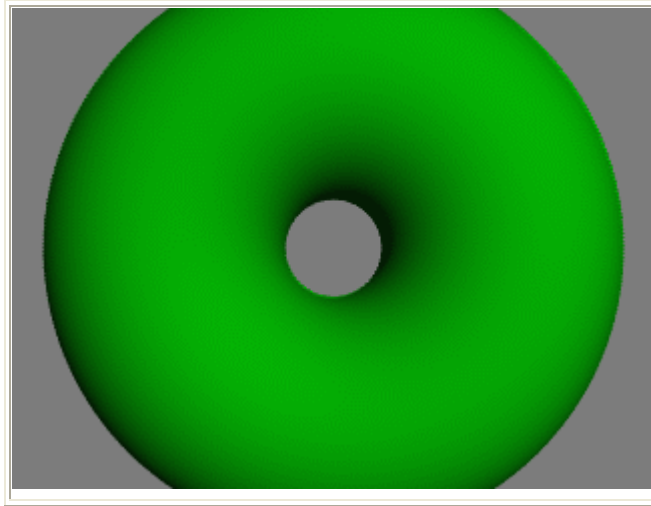



Fig. 54-Toro

Ehi ! Una ciambella con un serio problema di peso !
Con una sintassi così semplice, non c'è molto altro che possiamo fare con un toro a parte cambiargli la texture...o si ? Vediamo... i tori sono oggetti molto utili in CSG. Tentiamo un piccolo esperimento. Facciamo la differenza tra un toro ed un parallelepipedo :

```
difference {
torus { 4, 1
rotate x*-90 // così possiamo vederlo da sopra
}
box { <-5, -5, -1>, <5, 0, 1> }
pigment { Green }
}
```

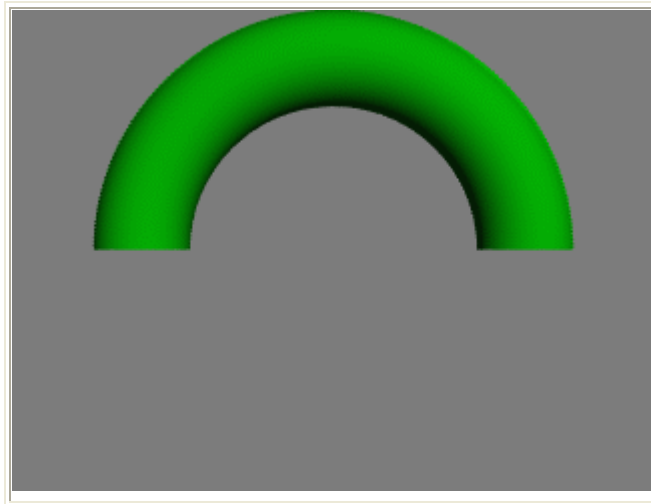



Fig. 55- Mezzo toro

Interessante...un mezzo toro. Ora ne aggiungiamo un altro ruotato dall'altra parte. Però, dichiariamo il primo mezzo toro e le trasformazioni necessarie in modo da poterlo riutilizzare :

```
#declare Half_Torus = difference { // Half Torus significa "mezzo
toro"
torus { 4, 1
rotate -90*x // così possiamo vederlo da sopra
```

```
}
box { <-5, -5, -1>, <5, 0, 1> }
pigment { Green }
}

#declare Flip_It_Over = 180*x // Flip It Over significa
"capovolgilo"
#declare Torus_Translate = 8 // Il doppio del raggio maggiore
```

Ora creiamo un'unione dei due mezzi tori :

```
union {
object { Half_Torus }
object { Half_Torus
rotate Flip_It_Over
translate Torus_Translate*x
}
}
```

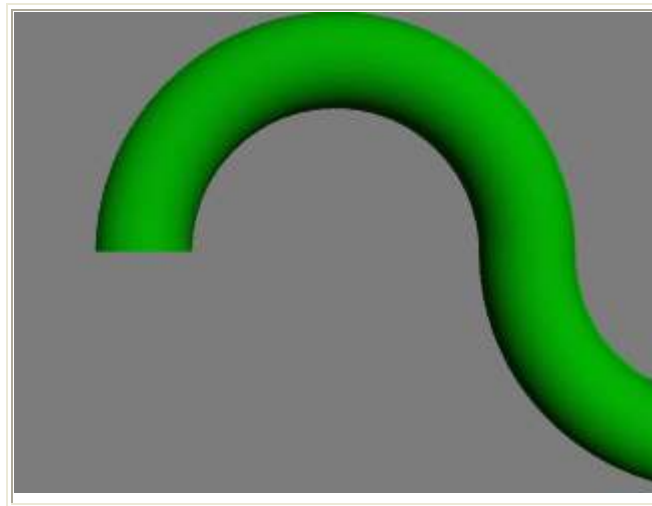



Fig. 56- I due mezzi tori sfalsati

Questo ci dà un oggetto a forma di S, ma non riusciamo a vederlo interamente dalla nostra posizione attuale. Aggiungiamo un altro po' di anelli, tre in ogni direzione, spostiamo l'oggetto in direzione +z e ruotiamolo in direzione -y in modo da poterne vedere di più. Notiamo anche che pare esserci un vuoto nel punto dove i mezzi tori si incontrano. Questo è dovuto al fatto che stiamo osservando la scena giacendo sul piano x-z. Cambieremo la coordinata y della macchina fotografica da 0 a 0.1 in modo da eliminare questo inconveniente.

```
union {
object { Half_Torus }
object { Half_Torus
rotate Flip_It_Over
translate x*Torus_Translate
}
object { Half_Torus
translate x*Torus_Translate*2
}
object { Half_Torus
rotate Flip_It_Over
```

```
translate x*Torus_Translate*3
}
object { Half_Torus
rotate Flip_It_Over
translate -x*Torus_Translate
}
object { Half_Torus
translate -x*Torus_Translate*2
}
object { Half_Torus
rotate Flip_It_Over
translate -x*Torus_Translate*3
}
object { Half_Torus
translate -x*Torus_Translate*4
}
rotate y*45
translate z*20
}
```

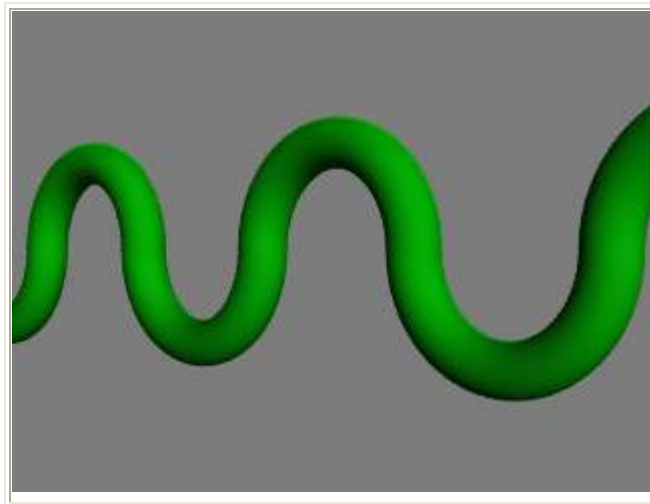



Fig. 57- "Serpente" formato da mezzi tori

Renderizzando questa scena vediamo qualcosa di ondulante, serpentiforme, o qualcosa del genere. Carino. Ma vogliamo fare qualcosa di utile, qualcosa che possiamo vedere nella vita reale. Che ne diresti di una catena ? Pensandoci un momento ci accorgiamo che un singolo anello di una catena può essere facilmente realizzato usando due mezzi tori e due cilindri per unire le due metà. Creiamo un nuovo file ed usiamo le stesse impostazioni di macchina fotografica, sfondo, sorgenti luminose e oggetti dichiarati che abbiamo usato in **tordemo.pov**.

```
#include "colors.inc"
camera {
location <0, .1, -40>
look_at 0
angle 30
}

background { color Gray50 }
light_source{ <300, 300, -1000> White }
#declare Half_Torus = difference {
```

```
torus { 4,1
sturm
rotate x*-90 // così possiamo vederlo da sopra
}
box { <-5, -5, -1>, <5, 0, 1> }
pigment { Green }
}

#declare Flip_It_Over = x*180
#declare Torus_Translate = 8
```

Ora, facciamo un toro completo usando le due metà :

```
union {
object { Half_Torus }
object { Half_Torus rotate Flip_It_Over }
}
```

Questa può sembrare una maniera inutile per fare un toro competo, ma in effetti stiamo per allontanare le due metà in modo da fare spazio per i cilindri. Per prima cosa, aggiungiamo prima del blocco `union` il cilindro dichiarato :

```
#declare Chain_Segment = cylinder { <0, 4, 0>, <0, -4, 0>, 1
pigment { Green }
}
```

Poi aggiungiamo due segmenti di catena all'unione e li trasliamo in modo che si allineino col raggio minore del toro su entrambi i lati :

```
union {
object { Half_Torus }
object { Half_Torus rotate Flip_It_Over }
object { Chain_Segment translate x*Torus_Translate/2 }
object { Chain_Segment translate -x*Torus_Translate/2 }
}
```

Ora trasliamo i due mezzi tori di +y e -y in modo che le estremità tronche si incontrino con le estremità dei cilindri. Questa distanza è uguale alla metà del valore `Torus_Translate` precedentemente dichiarato :

```
union {
object { Half_Torus
translate y*Torus_Translate/2
}
object { Half_Torus
rotate Flip_It_Over
translate -y*Torus_Translate/2
}
object { Chain_Segment
translate x*Torus_Translate/2
}
object { Chain_Segment
```

```
translate -x*Torus_Translate/2
}
}
```

Renderizziamo questa scena e voilà !

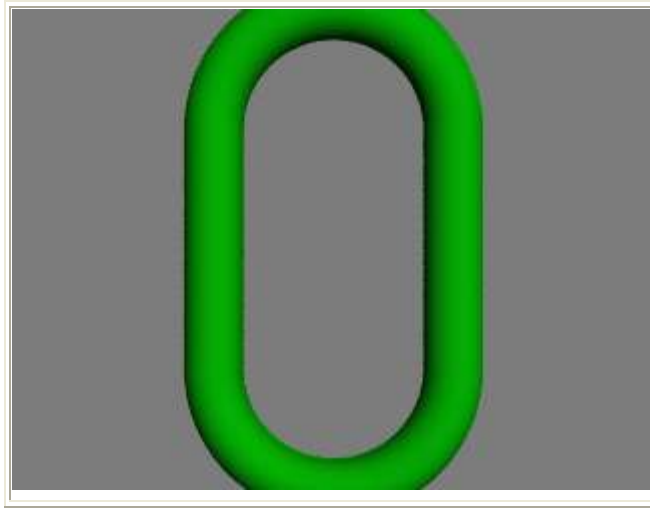



Fig. 58-Anello di una catena

Un anello di una catena. Ma non abbiamo ancora finito ! Chi ha mai sentito parlare di una catena verde ? Preferiremmo invece usare un bel colore metallico. Per prima cosa, eliminiamo tutti i blocchi `pigment` nelle dichiarazioni di tori e cilindri. Poi aggiungiamo queste righe prima di `union` :

```
#declare Chain_Gold = texture {
pigment { BrightGold }
finish {
ambient .1
diffuse .4
reflection .25
specular 1
metallic
}
}
```

E quindi aggiungiamo la texture all'unione e dichiariamo l'unione come un singolo anello :

```
#declare Link = union {
object { Half_Torus
translate y*Torus_Translate/2
}
object { Half_Torus
rotate Flip_It_Over
translate -y*Torus_Translate/2
}
object { Chain_Segment
translate x*Torus_Translate/2
}
object { Chain_Segment
```

```
translate -x*Torus_Translate/2
}
texture { Chain_Gold }
}
```

Ora facciamo un'unione di due anelli. Il secondo dovrà essere traslato di +y in modo che la sua parete interna si incontri appena con la parete interna dell'altra maglia, proprio come gli anelli di una catena. Questa distanza è il doppio della distanza `Torus_Translate` meno 2 (due volte il raggio minore). Si può descrivere con l'espressione :

```
Torus_Translate*2-2*y
```

E noi dichiariamo l'espressione come segue :

```
#declare Link_Translate= Torus_Translate*2-2*y
```

Nella parte relativa all'oggetto, useremo questo blocco per moltiplicarlo in modo da creare altri anelli. Ora, ruotiamo il secondo anello di 90*y in modo che sia perpendicolare al primo, come gli anelli di una catena. Infine, scaliamo l'unione di 0.25 in modo da poter vedere l'intera cosa :

```
union {
object { Link }
object { Link translate y*Link_Translate rotate y*90 }
scale .25
}
```

Renderizziamo questa scena e vedremo un paio di anelli molto realistico.

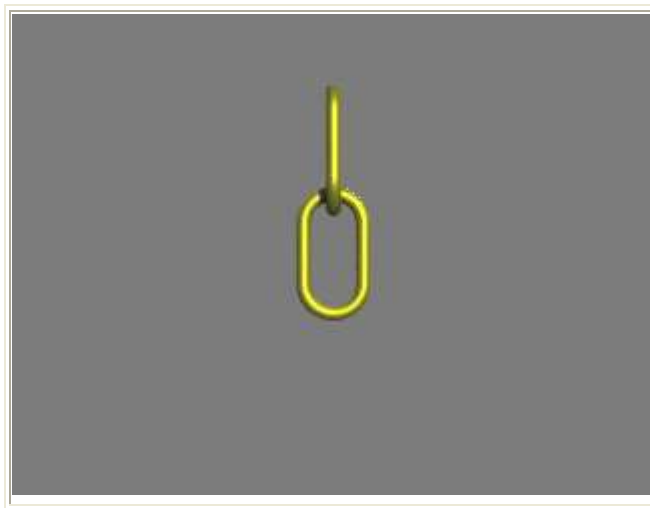



Fig. 59-Due anelli

Se vogliamo fare un'intera catena, dobbiamo dichiarare l'unione vista sopra e creare un'altra unione di questo oggetto dichiarato. Dobbiamo essere sicuri di rimuovere la frase `scale` dall'oggetto dichiarato :

```
#declare Link_Pair =
union {
object { Link }
object { Link translate y*Link_Translate rotate y*90 }
```

```
}
```

Ora dichiariamo la nostra catena :

```
#declare Chain = union {
object { Link_Pair}
object { Link_Pair translate y*Link_Translate*2 }
object { Link_Pair translate y*Link_Translate*4 }
object { Link_Pair translate y*Link_Translate*6 }
object { Link_Pair translate -y*Link_Translate*2 }
object { Link_Pair translate -y*Link_Translate*4 }
object { Link_Pair translate -y*Link_Translate*6 }
}
```

Ed infine, creiamo la nostra catena, con un paio di trasformazioni affinché sia più facile vederla. Queste includono rimpicciolirla di un fattore 10 e ruotarla in modo che si possano vedere bene tutti gli anelli.

```
object { Chain scale .1 rotate <0, 45, -45> }
```

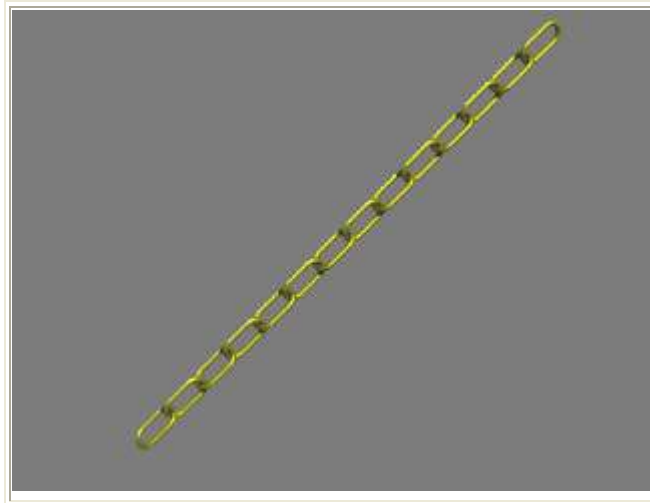



Fig. 60-I Tori possono essere usati per creare catene

Renderizziamo questa scena e dovremmo vedere una catena dorata molto realistica che si estende diagonalmente attraverso lo schermo

4.5 Geometria Solida Costruttiva (CSG)

Geometria Solida Costruttiva (in Inglese, Constructive Solid Geometry, CSG, N.d.T.) è un potente strumento per combinare oggetti primitivi e creare oggetti più complessi, come mostrato nei paragrafi seguenti.

4.5.1 Cosa Vuol Dire CSG ?

CSG significa Geometria Solida Costruttiva. POV-Ray ci permette di costruire solidi complessi combinando solidi primitivi in quattro modi diversi. Ci sono l'unione, dove due o più oggetti vengono sommati insieme, l'intersezione, dove due oggetti vengono combinati per ottenere un terzo oggetto che consiste del volume comune ai due oggetti di partenza. La differenza, dove gli oggetti successivi vengono sottratti dal primo, Infine, la fusione, che è come un'unione in cui le superfici interne vengono rimosse (utile in oggetti CSG trasparenti). Tratteremo in dettaglio ciascuno di questi nelle sezioni successive. Gli oggetti CSG possono essere estremamente complessi e

profondamente annidati. In altre parole, ci possono essere unioni di differenze, o intersezioni di fusioni, o differenze di intersezioni, o anche unioni di intersezioni di differenze di fusioni...ad infinitum. Gli oggetti CSG sono (quasi sempre) oggetti finiti che rispondono all'auto-bounding (vedi paragrafo "Controllo sul Bounding Automatico") e possono essere trasformati come qualunque altra primitiva di POV-Ray.

4.5.2 Unione

Proviamo a fare una semplice unione. Creiamo un file di nome **csgdemo.pov** e lo editiamo come segue :

```
#include "colors.inc"
camera {
location <0, 1, -10>
look_at 0
angle 36
}

light_source { <500, 500, -1000> White }
plane { y, -1.5
pigment { checker Green White }
}
```

Aggiungiamo due sfere traslate ciascuna di 0.5 unità sull'asse x nelle direzioni positiva e negativa. Le coloriamo una di blu e l'altra di rosso.

```
sphere { <0, 0, 0>, 1
pigment { Blue }
translate -0.5*x
}
sphere { <0, 0, 0>, 1
pigment { Red }
translate 0.5*x
}
```

Renderizziamo questo file a 200x150 -A.

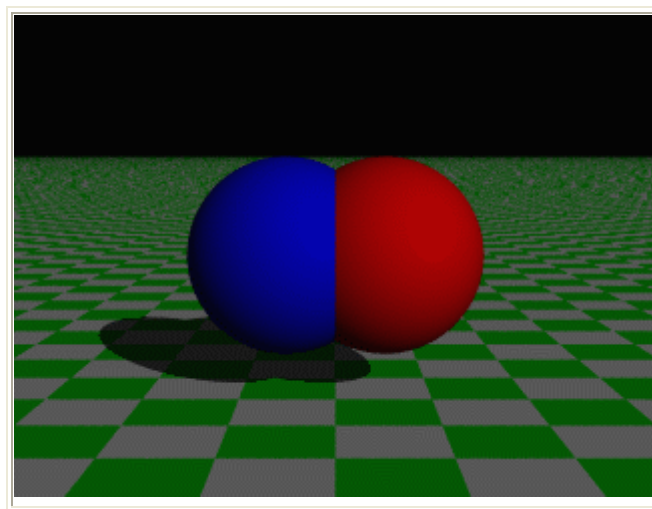



Fig. 61-Due sfere sovrapposte

Ora poniamo un blocco `union` attorno alle due sfere. Questo creerà una singola unione CSG dalle due sfere.

```
union{
sphere { <0, 0, 0>, 1
pigment { Blue }
translate -0.5*x
}
sphere { <0, 0, 0>, 1
pigment { Red }
translate 0.5*x
}
}
```

Renderizziamo nuovamente il file.

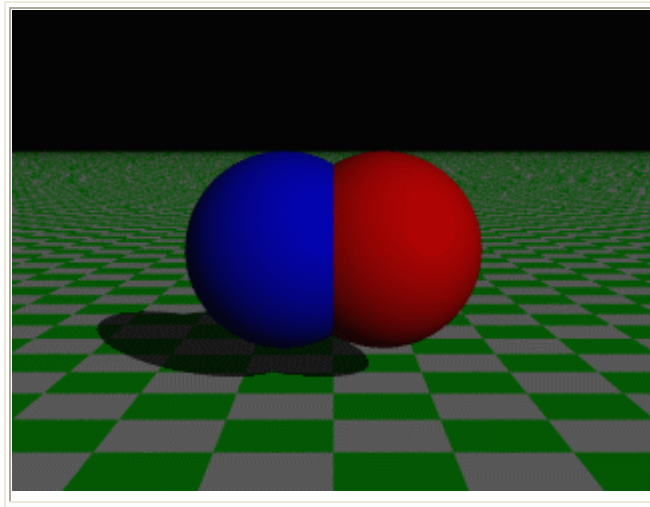



Fig. 62-L'unione di due sfere

L'unione non apparirà diversamente da prima, ma ora possiamo assegnare all'intera unione una singola texture e trasformarla come un unico oggetto. Proviamo :

```
union{
sphere { <0, 0, 0>, 1
translate -0.5*x*
}
sphere { <0, 0, 0>, 1
translate 0.5*x
}
pigment { Red }
scale <1, .25, 1>
rotate <30, 0, 45>
}
```

Renderizziamo nuovamente il file.

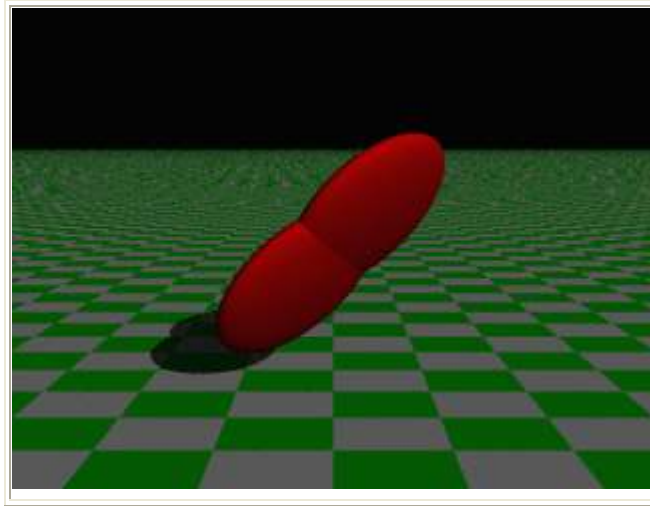



Fig. 63-Applicare le trasformazioni all'unione

Come possiamo vedere, l'oggetto è estremamente diverso. Sperimentiamo diversi valori di `scale` e `rotate` e proviamo qualche texture diversa.
Ci sono molti vantaggi nell'assegnare un'unica texture ad un oggetto CSG invece che assegnare la texture ad ogni componente individuale. Per prima cosa, è molto più semplice usare una sola texture se l'oggetto ha un gran numero di componenti perché se vogliamo cambiare il suo aspetto ci basta modificare una sola frase `texture`. Secondo, il file viene analizzato più velocemente dal programma dato che la texture viene messa in memoria una volta sola e richiamata dalle componenti dell'oggetto. Assegnare la stessa texture a tutte le n componenti di un oggetto CSG significa metterla in memoria n volte.

4.5.3 Intersezione

Adesso, usiamo le stesse sfere di prima per illustarre il secondo tipo di oggetti CSG, l'intersezione. Cambiamo la parola `union` in `intersection` e cancelliamo le frasi `scale` e `rotate` :

```
intersection {
sphere { <0, 0, 0>, 1
translate -0.5*x
}
sphere { <0, 0, 0>, 1
translate 0.5*x
}
pigment { Red }
}
```

Renderizziamo il file.

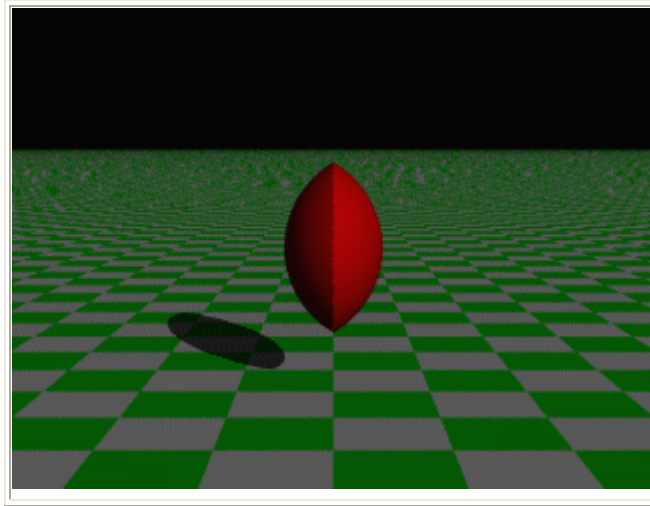



Fig. 64-Intersezione di due sfere

Vediamo un oggetto a forma lenticolare invece delle due sfere. Questo avviene perché un'intersezione consiste del volume compreso in entrambe le componenti, in questo caso la zona di spazio lenticolare si trova dove le due sfere si sovrappongono. Quest'oggetto ci piace e quindi lo useremo per studiare le differenze.

4.5.4 Differenza

Ruotiamo l'intersezione del paragrafo precedente attorno all'asse delle y in modo che la parte larga si trovi di fronte alla macchina fotografica.

```
intersection{
sphere { <0, 0, 0>, 1
translate -0.5*x
}
sphere { <0, 0, 0>, 1
translate 0.5*x
}
pigment { Red }
rotate 90*y
}
```

Creiamo un cilindro e piazziamolo nel centro della lente.

```
cylinder { <0, 0, -1> <0, 0, 1>, .35
pigment { Blue }
}
```

Renderizziamo la scena per vedere la posizione del cilindro.

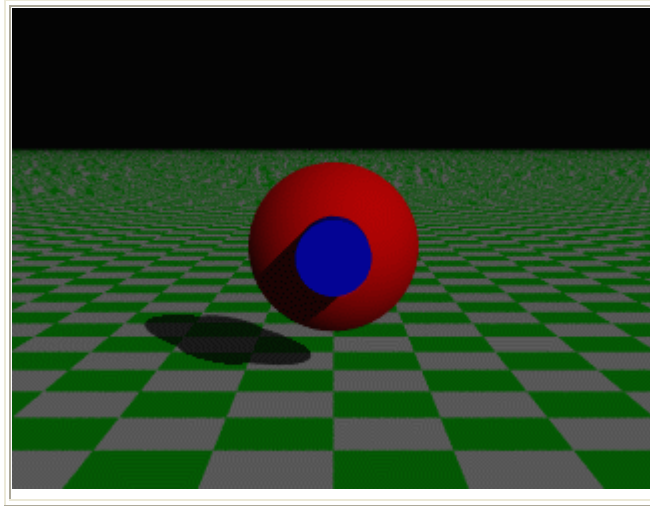



Fig. 65

Metteremo un blocco difference attorno all'intersezione ed al cilindro in questo modo :

```
difference {
intersection {
sphere { <0, 0, 0>, 1
translate -0.5*x
}
sphere { <0, 0, 0>, 1
translate 0.5*x
}
pigment { Red }
rotate 90*y
}
cylinder { <0, 0, -1> <0, 0, 1>, .35
pigment { Blue }
}
}
```

Renderizziamo nuovamente il file.

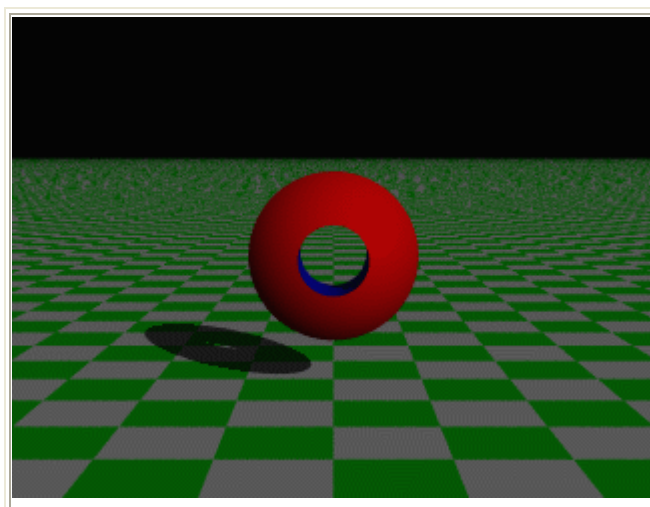



Fig. 66-Differenza

Vediamo l'intersezione con un foro nel mezzo, dove si trovava il cilindro. Il cilindro è stato sottratto dall'intersezione. Nota che il colore del cilindro fa sì che la superficie del buco sia colorata in blu. Se eliminiamo questo colore nella frase `cylinder` la superficie del buco sarà blu.
OK, andiamo avanti. Dichiariamo la nostra lente perforata in modo da assegnarle un nome. Eliminiamo anche tutte le texture perché vogliamo specificarle solo nell'unione finale.

```
#declare Lens_With_Hole = difference {
intersection {
sphere { <0, 0, 0>, 1
translate -0.5*x
}
sphere { <0, 0, 0>, 1
translate 0.5*x
}
rotate 90*y
}
cylinder { <0, 0, -1> <0, 0, 1>, .35 }
}
```

Usiamo un oggetto unione per costruire un oggetto complesso composto di copie della 'lente' :

```
union {
object { Lens_With_Hole translate <-.65, .65, 0> }
object { Lens_With_Hole translate <.65, .65, 0> }
object { Lens_With_Hole translate <-.65, -.65, 0> }
object { Lens_With_Hole translate <.65, -.65, 0> }
pigment { Red }
}
```

Renderizziamo la scena.

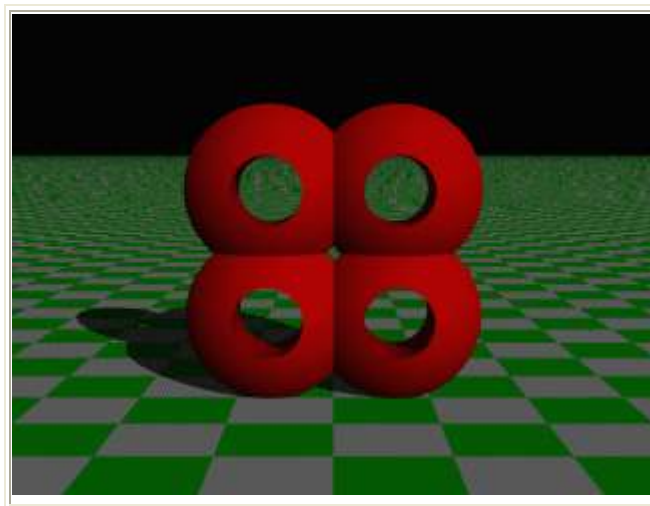



Fig. 67-Unione delle differenze

Un oggetto sicuramente interessante. Ma proviamo a spingerci oltre. Rendiamolo parzialmente trasparente aggiungendo una componente di filtro al blocco `pigment`.

```
union {
object { Lens_With_Hole translate <-.65, .65, 0> }
```

```
object { Lens_With_Hole translate <.65, .65, 0> }
object { Lens_With_Hole translate <-.65, -.65, 0> }
object { Lens_With_Hole translate <.65, -.65, 0> }
pigment { Red filter .5 }
}
```

Renderizziamo nuovamente il file.

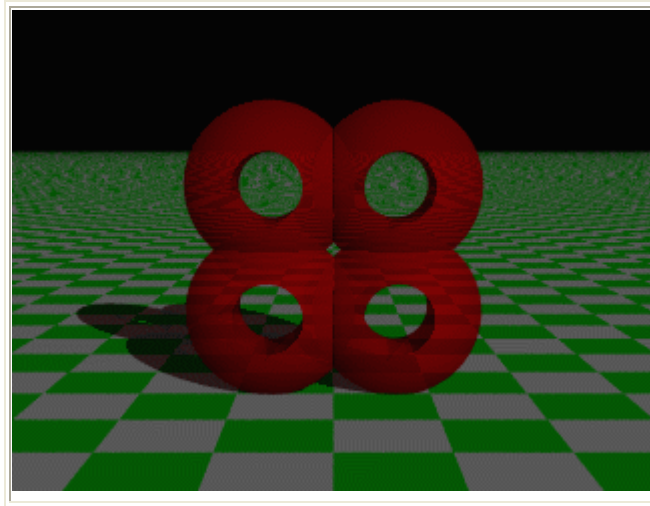



Fig. 68-Come sopra, ma trasparente

Questo è abbastanza bello, ma... possiamo vedere parti di ciascuno degli oggetti 'lente' all'interno dell'unione. Questo non va bene.

4.5.5 Fusione

Questo ci porta al quarto tipo di oggetto CSG, la 'fusione' (merge). Un oggetto fusione ha la stessa forma dell'oggetto unione, ma la geometria degli oggetti che si trovano all'interno della superficie non viene renderizzata. Questo dovrebbe eliminare il problema che avevamo. Proviamo :

```
merge {
object { Lens_With_Hole translate <-.65, .65, 0> }
object { Lens_With_Hole translate <.65, .65, 0> }
object { Lens_With_Hole translate <-.65, -.65, 0> }
object { Lens_With_Hole translate <.65, -.65, 0> }
pigment { Red filter .5 }
}
```

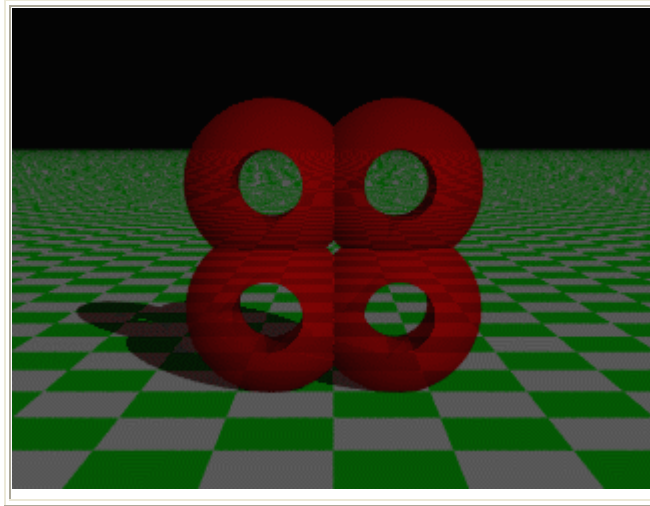



Fig. 69-Fusione

Funziona !

4.5.6 Tranelli in CSG

Un importante aspetto del codice degli oggetti CSG al quale dobbiamo fare molta attenzione.

4.5.6.1 Superfici Coincidenti.

POV-Ray usa test di superficie per determinare i punti in cui un raggio interseca un oggetto CSG. Un problema sorge quando le superfici di due differenti oggetti coincidono, dato che non c'è modo di distinguere (in base a problemi di calcolo) se un punto della superficie coincidente appartiene ad un oggetto o ad un altro. Vediamo il seguente esempio dove viene usato un cilindro per ricavare un buco da una scatola più grande.

```
difference {
box { -1, 1 pigment { Red } }
cylinder { -z, z, 0.5 pigment { Green } }
}
```

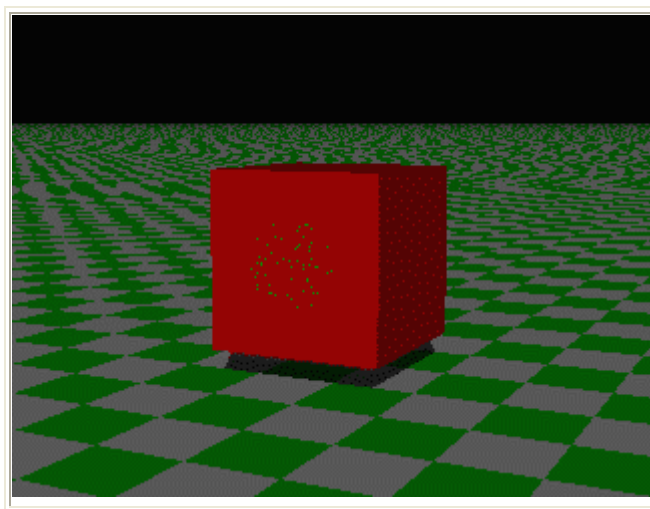



Fig. 70-Errore nel calcolo della differenza

renderizziamo quest'oggetto vediamo delle 'scaglie' rosse dove dovrebbe esserci il buco. Questo è causato dalle superfici coincidenti del cilindro e del cubo. A volte, viene colpita prima la superficie del cilindro dal raggio, dando un' immagine corretta del foro, a volte viene colpita prima la

superficie della scatola, dando un risultato sbagliato, in cui sparisce il buco ed appaiono le scaglie rosse. Questo problema può essere risolto aumentando leggermente le dimensioni del cilindro in modo da sbarazzarsi delle superfici coincidenti, in questo modo :

```
difference {
box { -1, 1 pigment { Red } }
cylinder { -1.001*z, 1.001*z, 0.5 pigment { Green } }
}
```

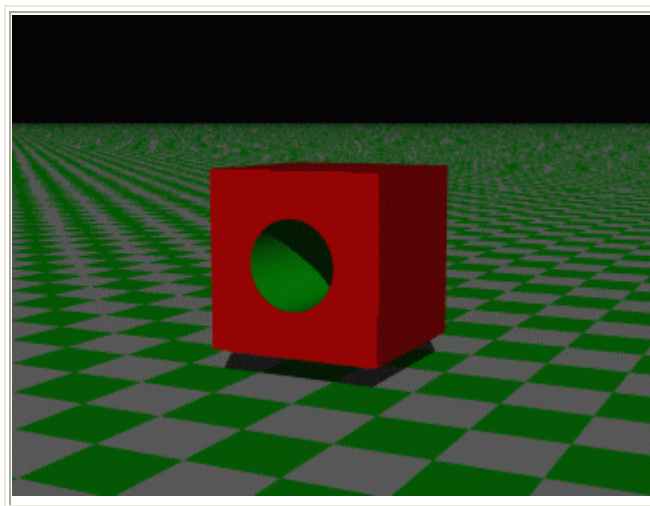



Fig. 71-Differenza corretta

In generale, dobbiamo rendere l'oggetto che viene sottratto un po' più grande, quando lo usiamo in una differenza di oggetti. Dobbiamo solo controllare la presenza di superfici coincidenti e aumentare appropriatamente le dimensioni dell'oggetto che viene sottratto per eliminare il

4.6 La Sorgente Luminosa

Una qualunque scena renderizzata con tecniche di raytracing (è il caso di POV-Ray), la luce necessaria per illuminare i nostri oggetti e le loro superfici deve provenire da una sorgente luminosa. Ci sono molti tipi di sorgente luminosa in POV-Ray e un uso attento del tipo giusto può condurre a risultati molto notevoli. Soffermiamoci un momento sui vari tipi di sorgente luminosa e sui loro vari parametri.

4.6.1 La Luce Ambiente

Luce ambiente è utilizzata per simulare l'effetto della riflessione interdiffusa. Se non ci fosse la riflessione interdiffusa, tutte le zone non illuminate direttamente da una sorgente luminosa, sarebbero completamente buie. POV-Ray utilizza la parola chiave `ambient` per determinare quanta luce proveniente dalla sorgente di 'luce ambiente' sia riflessa da una superficie.

Per valore predefinito, la luce ambiente, che emette luce in ogni direzione ed in ogni punto, è completamente bianca (rgb<1,1,1>). Cambiando il suo colore si possono ottenere alcuni effetti interessanti. Per prima cosa, il livello di illuminazione totale della scena può essere regolato con facilità. Invece di modificare tutte le frasi `ambient` di tutti i `finish` si modifica il valore della luce ambiente. Assegnando valori diversi, possiamo creare effetti interessanti, come un'illuminazione rossastra. Per maggiori dettagli riguardo la luce ambiente, vedi "Luce Ambiente". Sotto, un esempio di luce ambiente rossa

```
global_settings { ambient_light rgb<1, 0, 0> }
```

4.6.2 La Luce Puntiforme.

Una sorgente di luce puntiforme è esattamente ciò che il suo nome indica. Questa luce non ha dimensioni, è invisibile ed illumina tutto nella scena in maniera uguale, non importa quanto lontano dalla luce (ma questo comportamento può essere cambiato). Questa è la luce più semplice. Ci sono solo due parametri importanti, la posizione ed il colore. Creiamo una semplice scena ed inseriamoci una luce puntiforme. Creiamo un nuovo file e lo chiamiamo **litedemo.pov.** Lo editiamo come segue :

```
#include "colors.inc"
#include "textures.inc"

camera {
location <-4, 3, -9>
look_at <0, 0, 0>
angle 48
}
```

Aggiungiamo i seguenti, semplici oggetti :

```
plane { y, -1
texture {
pigment {
checker
color rgb<0.5, 0, 0>
color rgb<0, 0.5, 0.5>
}
finish {
diffuse 0.4
ambient 0.2
phong 1
phong_size 100
reflection 0.25
}
}
}

torus { 1.5, 0.5
texture { Brown_Agate }
rotate <90, 160, 0>
translate <-1, 1, 3>
}

box { <-1, -1, -1>, <1, 1, 1>
texture { DMFLightOak }
translate <2, 0, 2.3>
}

cone { <0,1,0>, 0, <0,0,0>, 1
texture { PinkAlabaster }
scale <1, 3, 1>
translate <-2, -1, -1>
}

sphere { <0,0,0>,1
```

```
texture { Sapphire_Agate }
translate <1.5, 0, -2>
}
```

E ora, una luce puntiforme :

```
light_source {
<2, 10, -3>
color White
}
```

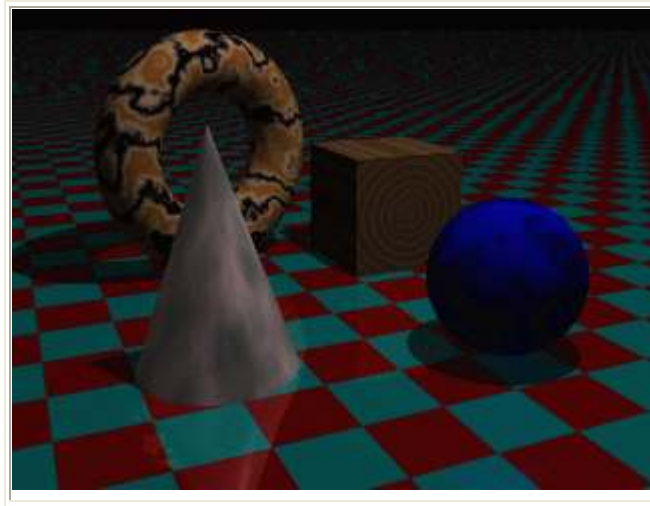



Fig. 72-Illuminazione della luce puntiforme

Renderizziamo questa scena a `200x150 -A` e vediamo che gli oggetti sono chiaramente visibili, con ombre ben nette. I lati degli oggetti curvi più vicini alla luce sono i più chiari, mentre le zone rivolte dalla parte opposta alla luce, sono le più scure. Notiamo anche che il piano a scacchiera è illuminato uniformemente fino all'orizzonte. Questo ci permette di vedere il piano, ma non è molto realistico.

4.6.3 La Luce Spot

Gli spot sono un tipo molto utile di sorgente luminosa. Possono essere utilizzati per aggiungere riflessi ed illuminare dettagli esattamente come agisce un fotografo per ottenere lo stesso risultato. Gli spot hanno un po' più parametri che non le luci puntiformi. Questi sono : `radius`, `falloff`, `tightness` e `point_at` (raggio, attenuazione, larghezza e bersaglio). Il parametro `raggio` rappresenta l'angolo del cono pienamente illuminato. Il parametro `falloff` rappresenta l'angolo del cono d'ombra dove la luce decade fino all'oscurità. Il parametro `point_at` indica la posizione a cui punta lo spot. Modifichiamo la luce nella nostra scena come segue :

```
light_source {
<0, 10, -3>
color White
spotlight
radius 15
falloff 20
tightness 10
point_at <0, 0, 0>
}
```

Renderizziamo questa scena a `200x150 -A`.

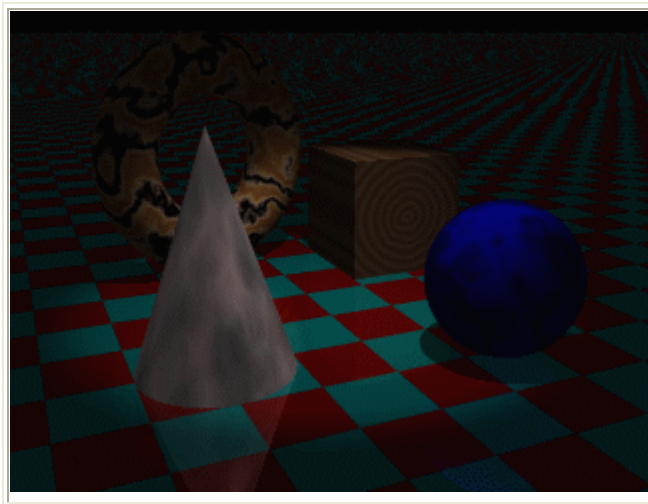



Fig. 73-Illuminazione con uno spot

Vediamo che solo gli oggetti sono illuminati. Il resto del piano e le parti più esterne degli oggetti sono al buio. C'è una larga zona in cui la luce declina dalla piena potenza al buio, ma le ombre sono ancora nettissime. Proviamo a giocare con qualcuno di questi parametri per vedere cosa fanno. Cambiamo il valore di `falloff` a 16 (deve sempre essere maggiore di `radius`) e renderizziamo di nuovo.

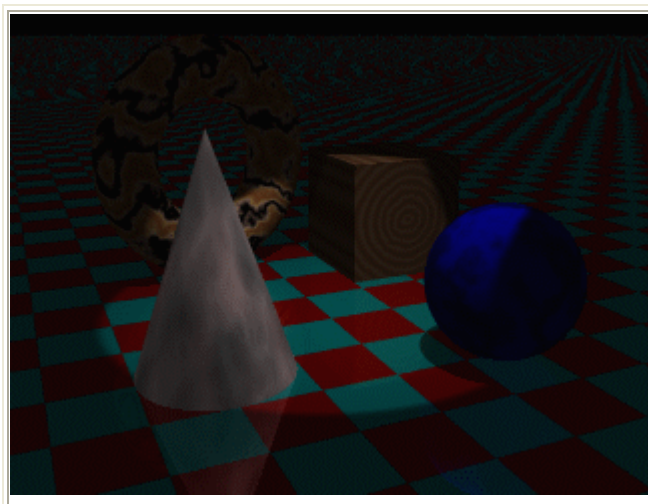



Fig. 74-Aumentare il valore di falloff

Ora la zona di `falloff` è molto stretta e gli oggetti sono completamente illuminati o completamente al buio. Ora riportiamo a 20 il valore del `falloff` e cambiamo il valore di `tightness` a 100 (maggiore = più stretto) e renderizziamo di nuovo.

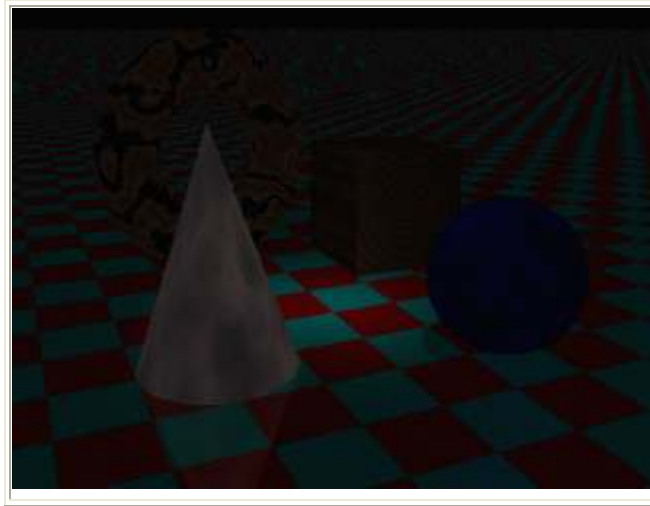



Fig. 75-Aumentare il valore di tightness

La luce sembra essersi ristretta, ma ciò che è veramente successo è che la zona di `falloff` è diventata talmente larga che il raggio appare effettivamente minore. Decidiamo che un valore di 10 (il valore predefinito) per `tightness` e 18 per `falloff` sono una regolazione ottimale per questo spot e ora vogliamo metterne qualcuno qui e là per la scena per aumentare l'effetto. Creiamo uno spot blu, leggermente più stretto di questo ed uno rosso, in aggiunta a quello bianco che abbiamo già :

```
light_source {
<10, 10, -1>
color Red
spotlight
radius 12
falloff 14
tightness 10
point_at <2, 0, 0>
}

light_source {
<-12, 10, -1>
color Blue
spotlight
radius 12
falloff 14
tightness 10
point_at <-2, 0, 0>
}
```

Renderizzando questa scena.

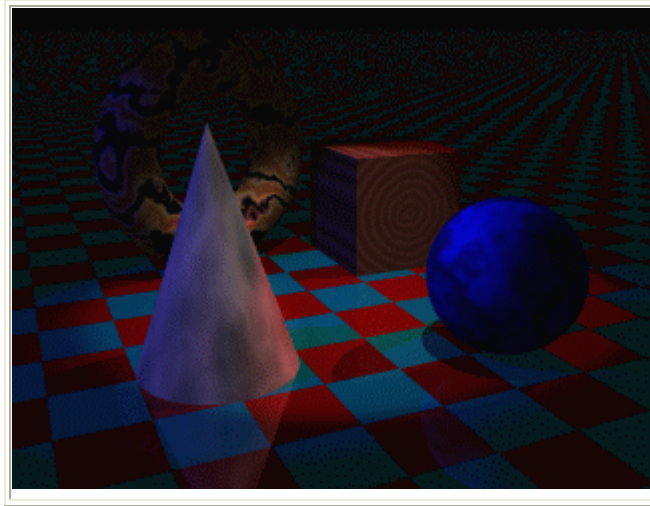



Fig. 76-Spot colorati

Vediamo che le abbiamo aggiunto un aspetto meravigliosamente misterioso. I tre spot convergono tutti sugli oggetti rendendoli blu su un lato e rossi sull'altro, con abbastanza bianco al centro per bilanciare l'effetto.

4.6.4 Luci Cilindriche

Gli spot sono conici, nel senso che il loro effetto cambia con la distanza. Più lontano si trova un oggetto dallo spot, più largo sarà il raggio della zona illuminata. Ma noi potremmo desiderare che raggio e `falloff` siano di una particolare dimensione indipendentemente dalla distanza. Per questo scopo, abbiamo bisogno delle luci cilindriche. Una sorgente luminosa cilindrica è come uno spot, con la differenza che raggio e `falloff` rimangono invariati indipendentemente da quanto lontano dalla luce si trovi l'oggetto. La forma della luce è quindi un cilindro anziché un cono. Possiamo specificare una luce cilindrica sostituendo la parola `spotlight` con la parola `cylinder`. Proviamo ora con la nostra scena, sostituendo tutte e tre le luci spot con luci cilindriche e renderizzando di nuovo.

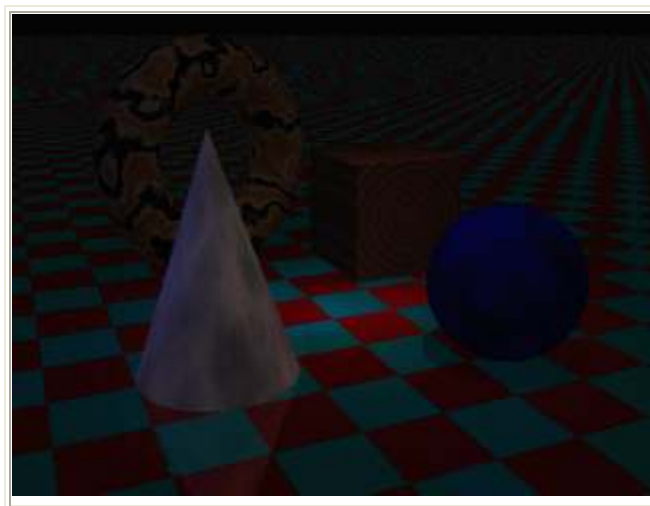



Fig. 77-Luce cilindrica

Vediamo che la scena è molto più scura. Questo avviene perché le luci cilindriche non permettono alla luce di diffondersi lateralmente come le luci spot. Abbiamo bisogno di valori maggiori di raggio e `falloff` per ottenere un risultato migliore. Tentiamo un raggio di 20 ed un valore di `falloff` di 30 per tutte e tre le luci.

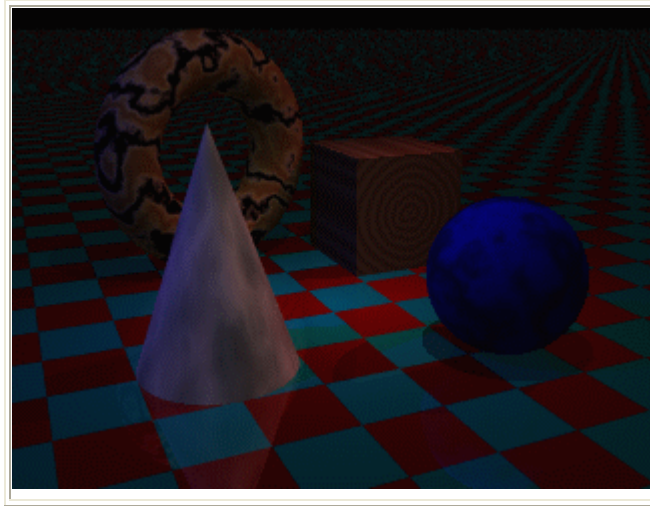



Fig. 78-Maggiori valori di raggio e falloff

Questa è la strada giusta !

4.6.5 Luci Diffuse

Fino ad ora tutte le nostre luci hanno una cosa in comune. Producono ombre nette. Questo avviene perché l'effettiva sorgente luminosa è un punto infinitamente piccolo. Gli oggetti si trovano direttamente nella luce, nel qual caso sono pienamente illuminati, oppure no, nel qual caso sono in ombra. Nella vita reale, questo tipo di illuminazione e di omreggiatura esiste solo nello spazio, dove la luce diretta del sole squarcia la totale oscurità dello spazio. Ma qui sulla Terra, la luce si piega attorno agli oggetti, rimbalza e normalmente la sorgente luminosa ha una qualche dimensione e può essere parzialmente nascosta alla vista (le ombre non sono più così nette). Esiste quella che viene chiamata penombra, un'area in cui non c'è la totale luce, né la penombra. Per simulare queste ombre leggere, un raytracer deve fornire una dimensione alla sorgente luminosa. POV-Ray esegue questo compito con una funzione chiamata `area_light`. Le dimensioni dell' `area_light` si estendono su due assi. Queste vengono specificate dai primi due vettori nella sintassi della luce. Dobbiamo anche specificare quante luci ci devono essere. Un alto numero di luci ci fornirà ombre sfumate, ma impiegherà più tempo per il rendering. Normalmente un insieme di 3*3 o 5*5 luci sarà sufficiente. Abbiamo anche la possibilità di specificare un valore di adattamento. La parola chiave `adaptive` dice al programma che può adattarsi alla situazione e tracciare solo i raggi necessari a determinare il valore del pixel. Se non viene usata la parola `adaptive` verrà calcolato un raggio per ogni luce dell'`area_light`. Questo può veramente rallentare le cose. Maggiore sarà il valore di adattamento, più 'pulita' risulterà l'ombra, ma ciò risulterà in un aumentare del tempo di rendering. Normalmente un valore di 1 è sufficiente. Infine, dovremmo probabilmente usare la parola chiave `jitter`. Questa dice a POV-Ray di spostare leggermente la posizione di ogni luce in modo da ottenere ombre veramente sfumate invece che fornirci un'ombra consistente di bande ravvicinate.

Bene, proviamone una. Commentiamo le luci cilindriche e aggiungiamo :

```
light_source {
10, -3>
White
area_light <5, 0, 0>, <0, 0, 5>, 5, 5
adaptive 1
jitter
}
```

Questa è una luce centrata in <2,10,-3>. E'larga 5 unità lungo l'asse x e 5 lungo l'asse z e contiene 25 (5*5) luci. Abbiamo specificato `adaptive 1` e `jitter`. Renderizziamo la scena a `200x150 -A`.

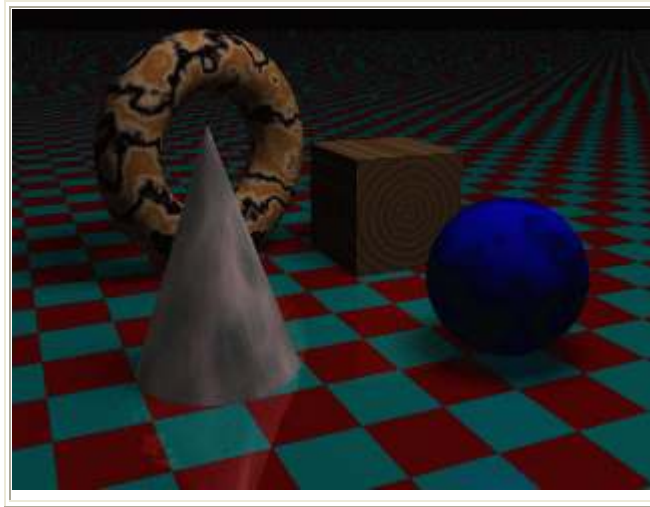



Fig. 79-Luci diffuse

Immediatamente, notiamo due cose. Il rendering è discretamente più lungo di quanto non fosse con una luce spot o puntiforme e le ombre non sono più nette ! Hanno una bella zona di penombra sfumata attorno. Aspetta, può migliorare. Anche gli spot e le luci cilindriche possono essere area lights. Hai presente quelle ombre nette che avevamo nella scena con gli spot ? Non sarebbe molto ragionevole usare un insieme di 5*5 luci per ottenere uno spot, ma un insieme più piccolo potrebbe fare un ottimo lavoro, dandoci una penombra giusta per uno spot. Proviamo. Commentiamo l'area light e cambiamo le luci cilindriche in modo che risultino :

```
light_source { <2, 10, -3>
color White
spotlight
radius 15
falloff 18
tightness 10
area_light <1, 0, 0>, <0, 0, 1>, 2, 2
adaptive 1 jitter point_at <0, 0, 0> }
light_source { <10, 10, -1>
color Red
spotlight
radius 12
falloff 14
tightness 10
area_light <1, 0, 0>, <0, 0, 1>, 2, 2
adaptive 1 jitter point_at <2, 0, 0> }
light_source { <-12, 10, -1> color Blue
spotlight
radius 12
falloff 14
```

```
tightness 10
area_light <1, 0, 0>, <0, 0, 1>, 2, 2 adaptive 1 jitter point_at
<-2, 0, 0> }
```

Abbiamo così tre spot 'allargati', quadrati di un'unità di lato, contenenti ciascuno una matrice di 2*2 luci, con tre colori diversi, che risplendono sulla nostra scena. Renderizziamo questa scena a `200x150 -A`.

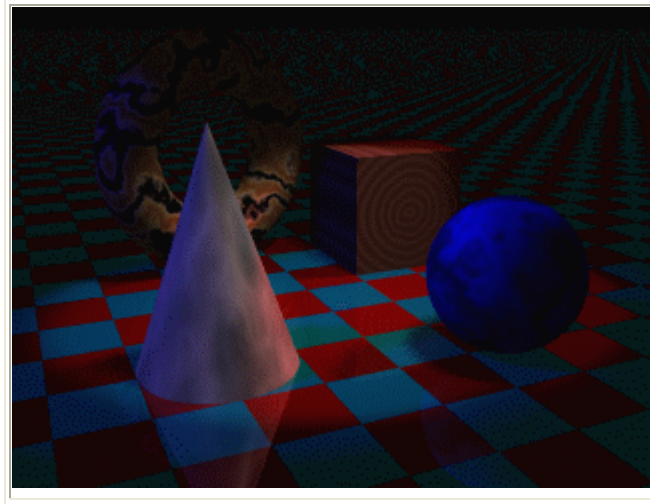



Fig. 80-Luci diffuse colorate

Sembra funzionare perfettamente. Tutte le nostre ombre hanno zone di penombra piccole e strette, proprio del genere che ci aspetteremmo di trovare su un oggetto illuminato da un vero spot.

4.6.6 Assegnare un Oggetto ad una Sorgente Luminosa

Le sorgenti luminose sono invisibili. Indicano solo il punto da cui si proietta la luce. Non hano una vera dimensione o una forma. Se vogliamo che la nostra luce sia visibile ed abbia una forma, possiamo usare la parola chiave `looks_like` (assomiglia). Possiamo cioè specificare che la nostra sorgente luminosa può assumere la forma di un qualunque oggetto scegliamo. Quando usiamo `looks_like` la parola chiave `no_shadow` (senza ombra) viene automaticamente applicata all'oggetto. Questo avviene per evitare che l'oggetto blocchi la luce proveniente dalla sorgente. Se vogliamo che ci sia dell'ombra, invece, è meglio usare un'unione per ottenere lo stesso risultato. Aggiungiamo un oggetto del genere alla nostra scena. Questa è una lampadina che abbiamo fatto proprio per questo scopo :

```
#declare Lightbulb = union {
merge {
sphere { <0,0,0>,1 }
cylinder { <0,0,1>, <0,0,0>, 1
scale <0.35, 0.35, 1.0>
translate 0.5*z
}
texture {
pigment {color rgb <1, 1, 1>}
finish {ambient .8 diffuse .6}
}
}
cylinder { <0,0,1>, <0,0,0>, 1
scale <0.4, 0.4, 0.5>
texture { Brass_Texture }
```

```
translate 1.5*z
}
rotate -90*x
scale .5
}
```

Ora aggiungiamo la sorgente luminosa :

```
light_source {
<0, 2, 0>
color White
looks_like { Lightbulb }
}
```

Renderizzando questa scena.

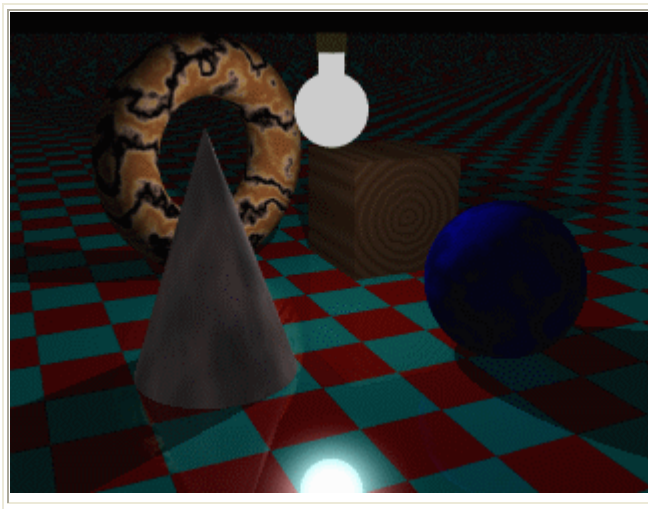



Fig. 81-Luce a forma di...

Vediamo che una lampadina discretamente credibile adesso illumina la scena. Comunque, se non specifichiamo un valore alto per `ambient`, la lampadina non è illuminata dalla sorgente luminosa. D'altra parte, tutte le ombre sono proiettate dalla lampadina, come dovrebbero realmente essere. Le ombre sono però troppo nette. Usiamo allora un'area light :

```
light_source {
<0, 2, 0>
color White
area_light <1, 0, 0>, <0, 1, 0>, 2, 2
adaptive 1
jitter
looks_like { Lightbulb }
}
```

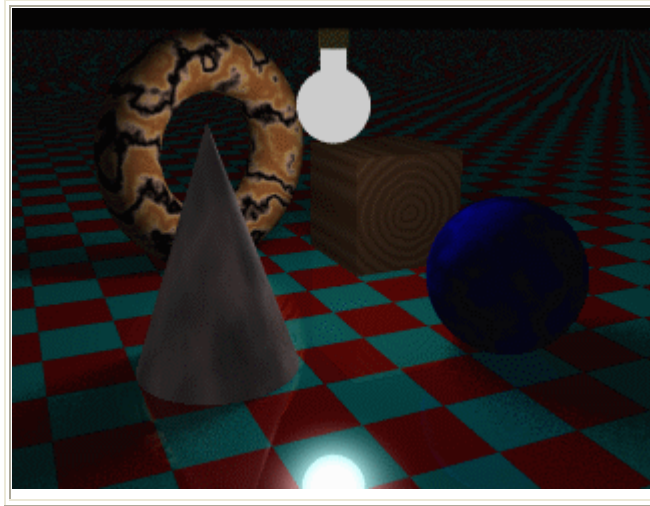



Fig. 82-Luci più attenuate

Notamo che abbiamo messo quest'area light nel piano x-y invece che nel piano x-z. Notiamo anche che l'aspetto della lampadina non è influenzato in alcun modo dalla sorgente luminosa. La lampadina deve essere illuminata da qualche altra sorgente lumnosa, o, come in questo caso, da un alto valore del parametro `ambient`. Risultati più interessanti possono essere ottenuti in questo caso usando le Halo (vedi paragrafo "Halo").

4.6.7 Funzioni Speciali

4.6.7.1 Luci Senza Ombra

Per non fare proiettare ombre alle sorgenti luminose, si può assegnare loro la parola chiave `shadowless`. Certe volte, le scene sono difficili da illuminare propriamente usando le luci che abbiamo scelto per illuminare i nostri oggetti. Applicare un alto valore del parametro `ambient` non è pratico e porta a risultati non realistici. Allora, posizioniamo un paio di luci di riempimento nella nostra scena. Le luci di riempimento sono semplicemente luci più deboli con la parola chiave `shadowless` che servono ad aumentare l'illuminazione delle zone della scena che possono non essere adeguatamente illuminate. Proviamo ad usarne una nella nostra scena. Ricordi i nostri tre spot 'area' ? torniamo indietro, togliamo il commento e commentiamo invece ogni altra luce che abbiamo inserito. Poi aggiungiamo il seguente testo :

```
light_source {
<0, 20, 0>
color Gray50
shadowless
}
```

Questa è una luce piuttosto debole che si trova 20 unità sopra il centro della scena. Fornirà una tenue illuminazione a tutti gli oggetti compreso il piano nello sfondo. Renderizziamo e vediamo.

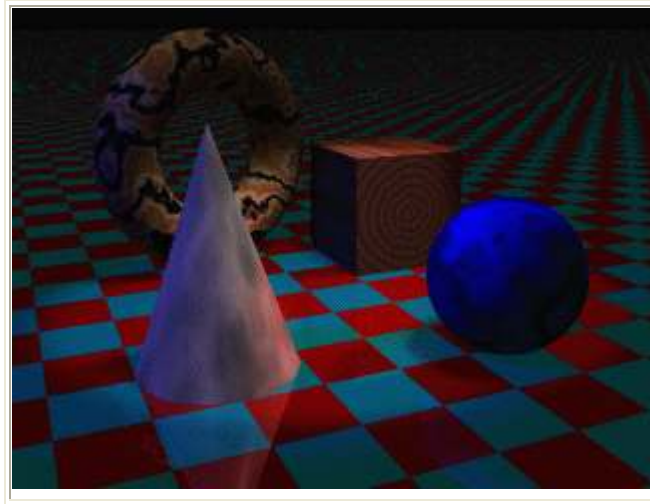



Fig. 83-Luci senza ombra

4.6.7.2 Attenuazione

Se è il realismo che vogliamo, non è realistico che il piano sia uniformemente illuminato fino all'orizzonte. Nella realtà, la luce viene dispersa lungo il suo cammino e perde quindi gradualmente la capacità di illuminare le cose via via più lontane dalla sorgente. Per simulare questo effetto, POV-Ray ci permette di usare due parole chiave : `fade_distance` e `fade_power`, un valore esponenziale che determina il tasso di attenuazione della luce. Applichiamo queste parole chiave alla nostra luce di riempimento. Per prima cosa, la rendiamo leggermente più brillante cambiandone il colore da `Gray50` a `Gray75`. Poi, la modifichiamo come segue :

```
light_source {
<0, 20, 0>
color Gray75
fade_distance 5
fade_power 1
shadowless
}
```

Che significa che la luce avrà (`fade_distance 5`) piena potenza fino a cinque unità di distanza dalla sorgente luminosa. Il valore `fade_power 1` significa che l'attenuazione varierà linearmente con la distanza. Renderizziamo la scena per vedere il risultato.

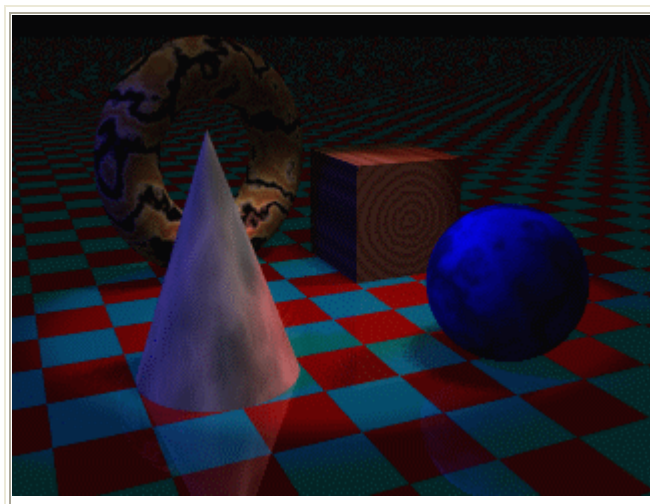



Fig. 84-Attenuazione

Ha funzionato ! Ora proviamo una `fade_power` di 2 ed una `fade_distance` di 10.

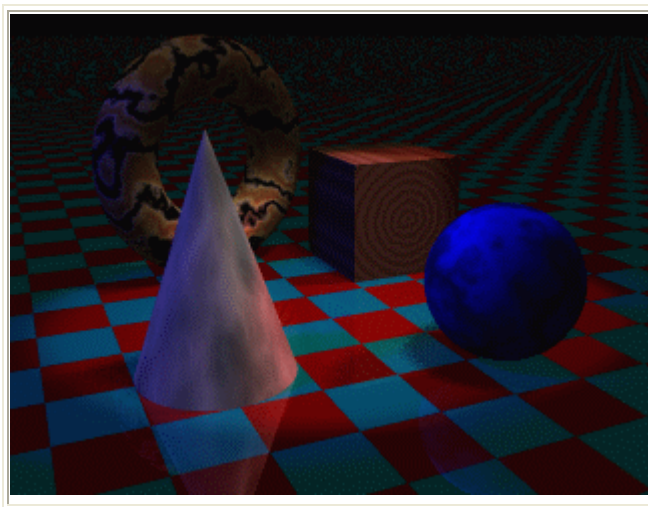



Fig. 85-Lavorare sull'attenuazione

Nuovamente, funziona bene ! Il raggio di attenuazione (`falloff)` è molto più piccolo con `fade_power` 2 per cui siamo costretti ad aumentare `fade_distance` al valore di 10.

4.6.7.3 Sorgenti Luminose ed Atmosfera

Per definizione, più che per incidente, le sorgenti luminose interagiscono con l'atmosfera, cioè la loro luce è dispersa dall'atmosfera. Questo effetto può essere disattivato aggiungendo `atmosphere off` alla frase `light_source`. La luce emessa da una sorgente luminosa, può anche essere attenuata dall'atmosfera (ed anche dalla nebbia) e quindi diminuirà durante il suo percorso, aggiungendo `atmospheric_attenuation on`. Il decadimento è esponenziale e dipende dal parametro `distance` dell'atmosfera (o della nebbia). E' da notare che questa funzione ha effetto solo sulla luce proveniente direttamente dalla sorgente luminosa. La luce riflessa e la luce rifratta non vengono modificate. Facciamo qualche esperimento con queste parole chiave. Per prima cosa dobbiamo aggiungere un'atmosfera alla nostra scena.

```
#include "atmos.inc"
atmosphere { Atmosphere2 }
```

Eliminiamo, commentandole, le tre linee che cambiano ciascuno dei tre spot in area light. Altrimenti il rendering sarebbe troppo lungo.

```
//area_light <1, 0, 0>, <0, 0, 1>, 2, 2
//adaptive 1
//jitter
```

Renderizzando la scena a `200x150 -A`

Fig. 86- atmosfera

vediamo che gli spot sono resi visibili. Possiamo vedere dove la luce rossa e quella blu si incrociano e dove quella bianca illumina il centro della scena. Notiamo anche che le luci diminuiscono in intensità man mano che la luce scende dalla sorgente agli oggetti. La luce rossa è scomparsa nella parte inferiore sinistra della scena e la luce blu è scomparsa nella parte inferiore destra. Questo è dovuto all'attenuazione atmosferica e fornisce ulteriore realismo alla scena. L'interazione atmosfera - sorgente luminosa, dà alla nostra scena un aspetto fumoso, misterioso, ma il rendering ha preso molto tempo. Cambiare gli spot in area light prolungherebbe ulteriormente il rendering. Questo è l'inevitabile compromesso tra velocità del rendering e qualità dell'immagine.

4.7 Semplici Opzioni sulle Texture

Le immagini che abbiamo renderizzato finora erano abbastanza noiose per quello che riguarda l'aspetto degli oggetti. Aggiungiamo qualche aspetto interessante alle texture.

4.7.1 Finitura della Superficie

Una delle funzioni principali di un ray-tracer è la sua capacità di fare cose interessanti con la finitura della superficie degli oggetti, come riflessione e illuminazione. Aggiungiamo un bel riflesso Phong (un punto luminoso) alla sfera. Per fare questo abbiamo bisogno di aggiungere la parola chiave `finish` seguita da un parametro. Modifichiamo la definizione della sfera in :

```
sphere { <0, 1, 2>, 2
texture {
pigment { color Yellow } // Yellow è predefinito in COLORS.INC
finish { phong 1 }
}
}
```

Renderizziamo la scena.

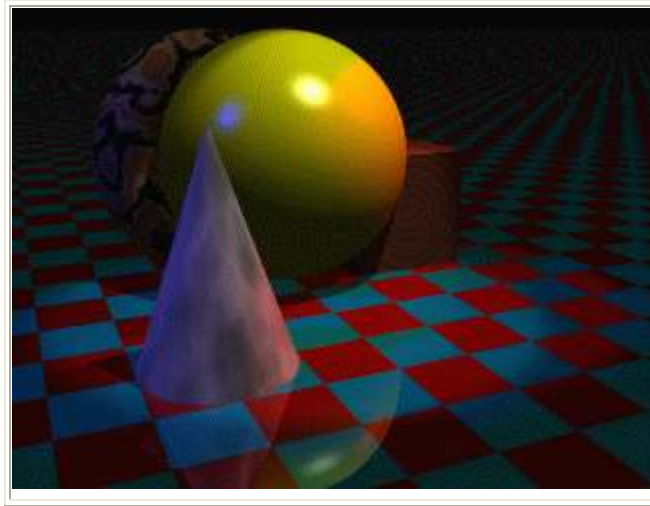



Fig. 87-Finitura

La parola chiave `phong` aggiunge un riflesso dello stesso colore della luce che brilla sull'oggetto. Aggiunge molta credibilità all'immagine e rende gli oggetti lisci e brillanti. Valori minori assegnati al parametro `phong` renderanno il riflesso meno brillante (i valori dovranno essere compresi tra 0 ed 1).

4.7.2 Aggiungere Rugosità

Il riflesso che abbiamo aggiunto ci mostra come buona parte della nostra percezione dipenda molto dalle proprietà di riflessione della luce di un oggetto. Il raytracing può aumentare questo effetto facendo degli scherzi alla nostra percezione e facendoci vedere dei complessi dettagli che in realtà non esistono. Supponiamo di volere una superficie molto irregolare per il nostro oggetto. Sarebbe molto difficoltoso modellare matematicamente centinaia di asperità sulla superficie. Possiamo comunque simulare l'aspetto di queste asperità modificando il modo in cui la luce si riflette sulla superficie dell'oggetto. I calcoli sulla riflessione dipendono da un vettore che viene chiamato *normale alla superficie*. Questo è un vettore che si alza perpendicolarmente alla superficie e che le è perpendicolare. Modificando artificialmente o perturbando questo vettore noi possiamo simulare le asperità. Modifichiamo la scena come segue e la renderizziamo :

```
sphere { <0, 1, 2>, 2
texture {
pigment { color Yellow }
normal { bumps 0.4 scale 0.2 }
finish { phong 1}
}
}
```

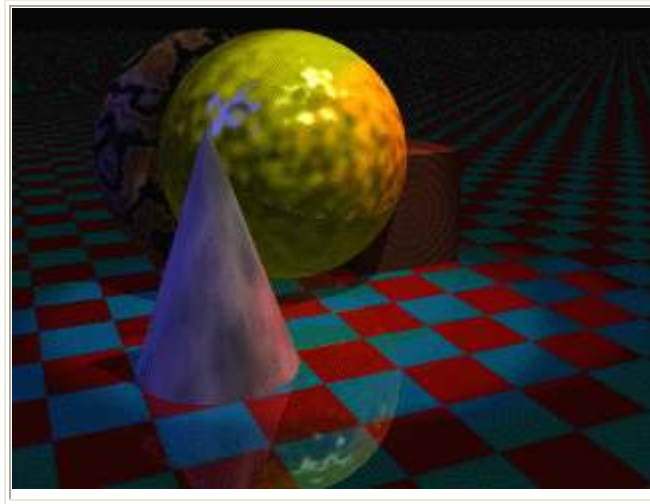



Fig. 88-Rugosità

Questo fa usare a POV-Ray un pattern di asperità per modificare la normale alla superficie. Il valore 0.4 controlla la profondità apparente delle asperità. Normalmente le asperità sono larghe circa un'unità, il che non funziona molto bene con una sfera di raggio 2 come la nostra, per cui l'istruzione `scale` restringe ad 1/5 le asperità ma non modifica la loro profondità.

4.7.3 Creare Pattern di Colore

Possiamo fare di più che assegnare un unico colore ad un oggetto. Possiamo creare complessi motivi nel blocco `pigment` come in questo esempio :

```
sphere { <0, 1, 2>, 2
texture {
pigment {
wood
color_map {
[0.0 color DarkTan]
[0.9 color DarkBrown]
[1.0 color VeryDarkBrown]
}
turbulence 0.05
scale <0.2, 0.3, 1>
}
finish { phong 1 }
}
}
```

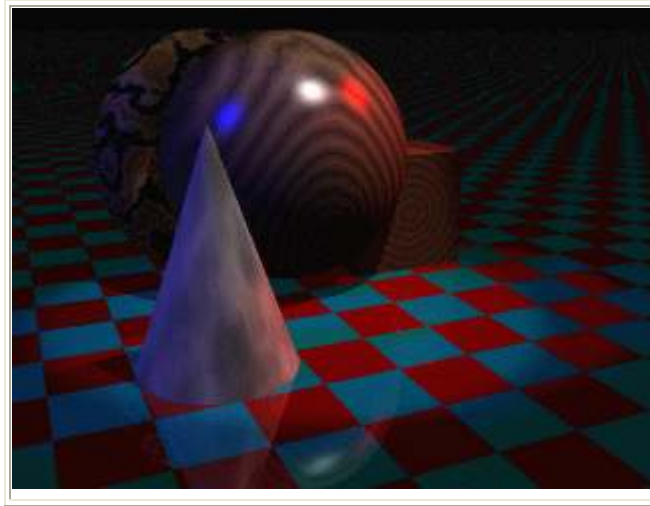



Fig. 89-Wood

La parola chiave `wood` specifica un motivo di pigmentazione formato da cerchi concentrici come gli anelli nel legno. La parola chiave `color_map` specifica che il colore del legno dovrebbe passare da `DarkTan` a `DarkBrown` nel primo 90% della venatura e da `DarkBrown` a `VeryDarkBrown` nel restante 10%. La parola chiave `turbulence` 'agita' un po' il motivo in modo che le venature non siano cerchi perfetti e la frase `scale` mette a punto la grandezza del motivo.
La maggior parte dei pattern sono impostati in modo da fornire un 'passaggio' su una sfera di raggio 1. Un 'passaggio' è definito in maniera molto grossolana come una transizione di colore. Per esempio, una texture 'legno' farebbe un solo anello se applicata (senza ridimensionarla) ad una sfera di raggio 1. In questo esempio ridimensioniamo il motivo usando il comando `scale` seguito da un vettore. In questo caso, lo abbiamo ridimensionato di 0.2 nella direzione delle x, 0.3 nella direzione delle y e 1 nella direzione delle z (cioè, rimane inalterato in quella direzione). Valori di `scale` maggiori di 1 'stireranno' un elemento. Valori minori di 1 lo restringeranno, valori di `scale` uguali ad uno lo lasceranno invariato.

4.7.4 Texture Predefinite

POV-Ray ha alcune texture molto sofisticate predefinite nei file .INC standard **glass.inc**, **metals.inc**, **stones.inc** e **woods.inc**. Alcune sono texture complete con tutti i parametri di pigment, `normal` e/o `finish` già definiti. Alcune sono solo pigmenti o solo finiture. Modifichiamo la definizione della nostra sfera e la renderizziamo di nuovo :

```
sphere { <0, 1, 2>, 2
texture {
pigment {
DMFWood4 // predefinita in textures.inc
scale 4 // ridimensiona della stessa quantità
// in tutte le direzioni
}
finish { Shiny } // predefinita in finish.inc
}
}
```

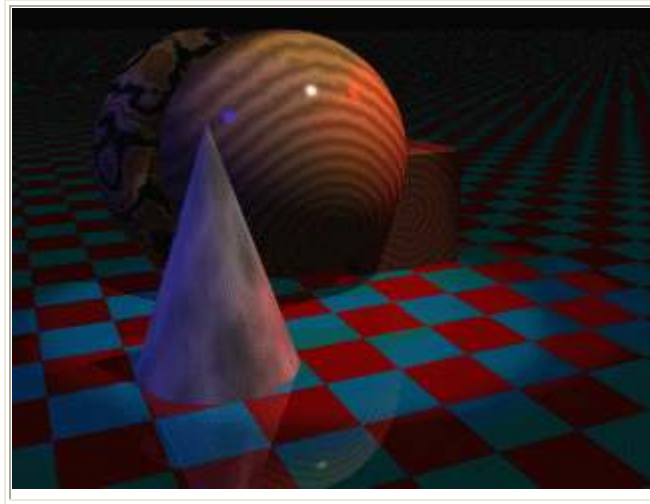



Fig. 90-Esempio di texture predefinita

L'identificatore di pigmento `DMFWood4` è già stato rimpicciolito molto quando è stato definito. Per questo esempio, vogliamo ingrandirlo. Dato che vogliamo ingrandirlo uniformemente possiamo mettere un singolo valore davanti alla parola `scale` invece che specificare un vettore composto dai fattori di ridimensionamento per gli assi x, y, z.
Guardiamo il file **textures.inc** per vedere quali texture e pigmenti sono definiti e li proviamo. Basta inserire il nome del nuovo pigmento dove si trovava `DMFWood4` o provare una finitura diversa al posto di `Shiny` e renderizzare di nuovo il nostro file. Ecco un esempio di come usare un identificatore di texture completo al posto delle sue parti :

```
sphere { <0, 1, 2>, 2
texture { PinkAlabaster }
}
```

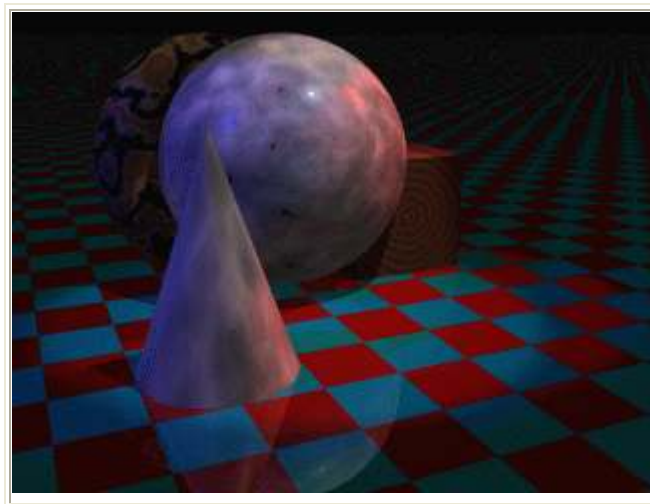



Fig. 91-Un'altra texture predefinita

4.8 Opzioni Avanzate sulle Texture

La capacità, estremamente potente, di POV-Ray di gestire le texture è una caratteristica che lo separa dagli altri programmi di raytracing. Fino ad ora non abbiamo provato niente di troppo complesso ma dovremmo essere abbastanza a nostro agio con la sintassi del programma per provare alcune delle opzioni più avanzate sulle texture. Ovviamente, non possiamo provarle tutte. Servirebbe un tutorial di molte più pagine per usare ogni opzione relativa alle texture disponibile in POV-Ray. Per questo tutorial, ci accontenteremo di provarne solo alcune per dare un'idea di come

vengono create le texture. Con un po' di pratica, presto saremo in grado di crearci le nostre (belle) texture personali.

4.8.1 Pattern sui Pigmenti e sulle Normali

Le versioni precedenti di POV-Ray facevano una distinzione tra i pattern da applicare al pigmento e quelli da applicare alle normali. Con POV-Ray 3.0 questa restrizione è stata rimossa, in modo che tutti i pattern elencati nel paragrafo "Motivi" possono essere utilizzati sia nelle frasi `pigment` che nelle frasi `normal`.

4.8.2 Pigmenti

Ogni superficie deve avere un colore. In POV-Ray questo colore viene chiamato `pigment` (pigmento). Non è necessario che questo sia un singolo colore. Può essere un motivo composto di più colori, una lista di colori od anche un'immagine. I pigmenti possono anche essere stratificati uno sopra l'altro fintanto che gli strati sovrastanti siano almeno in parte trasparenti in modo che quelli sottostanti possano essere visti. Divertiamoci con alcuni pigmenti.
Creiamo un file chiamato **texdemo.pov** e lo editiamo come segue :

```
#include "colors.inc"
camera {
location <1, 1, -7>
look_at 0
angle 36
}

light_source { <1000, 1000, -1000> White }
plane { y, -1.5
pigment { checker Green, White }
}

sphere { <0,0,0>, 1
pigment { Red }
}
```

Renderizzando velocemente questo file a `200x150 -A`

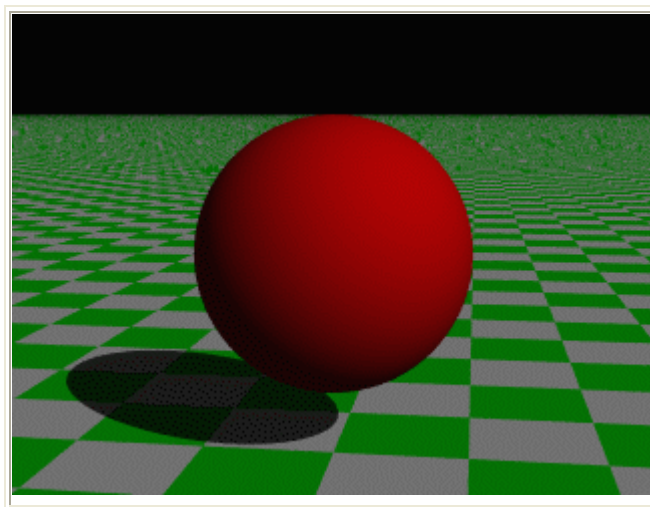



Fig. 92-Sfera rossa

vediamo che si tratta di una semplice sfera rossa contro un piano a scacchi verdi e bianchi. Utilizzeremo questa sfera come terreno di prova per le nostre texture.

4.8.2.1 Usare Liste di Colori

Prima di iniziare dovremmo notare che abbiamo già fatto un tipo di pigmento, la lista di colori. Nell'esempio precedente, abbiamo usato un motivo a scacchiera nel nostro piano. Ci sono altri due tipi di liste di colori, mattoni (`brick`) ed esagonale (`hexagon`). Proviamoli velocemente. Per prima cosa, cambiamo la frase `pigment` del piano come segue :

```
pigment { hexagon Green, White, Yellow }
```

Renderizzandolo vediamo un motivo esagonale a tre colori.

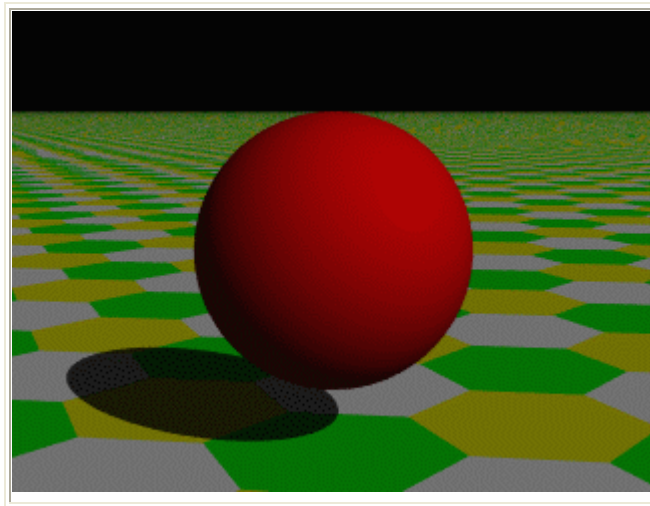



Fig. 93-Pavimento ad esagoni

Nota che questo motivo richiede appunto la specificazione di tre colori. Ora lo cambiamo di nuovo in...

```
pigment { brick Gray75, Red rotate -90*x scale .25 }
```

Guardando l'immagine risultante,

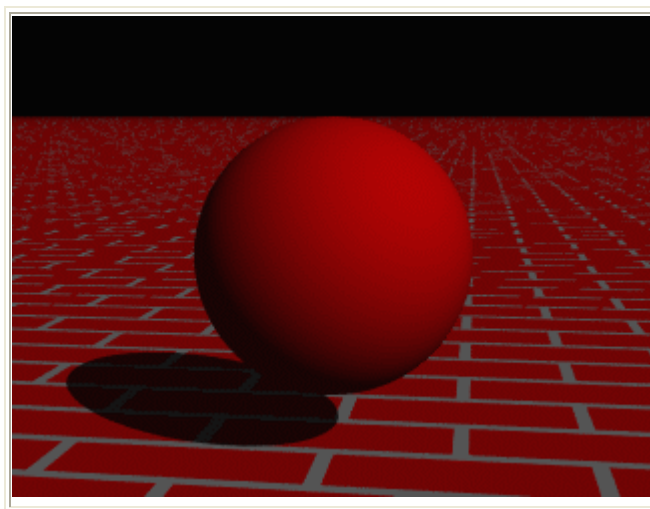



Fig. 94-Mattoni

Notiamo che il piano è ora decorato da un motivo a mattoni. Notiamo che abbiamo dovuto ruotare il motivo per farlo apparire correttamente sul piano. Questo motivo normalmente viene utilizzato su superfici verticali. Abbiamo dovuto anche rimpicciolire un po' il motivo in modo da poterlo vedere

più facilmente. Possiamo giocare con questi pigmenti a lista di colore, cambiare i colori, ecc. Fino ad ottenere un pavimento che ci piace.

4.8.2.2 Usare Pigmenti e Pattern

Iniziamo con l'assegnare una texture alla nostra sfera utilizzando un pattern ed una mappa colore consistente di tre colori. Sostituiamo il blocco `pigment` con il testo seguente :

```
pigment {
gradient x
color_map {
[0.00 color Red]
[0.33 color Blue]
[0.66 color Yellow]
[1.00 color Red]
}
}
```

Renderizzando questo esempio

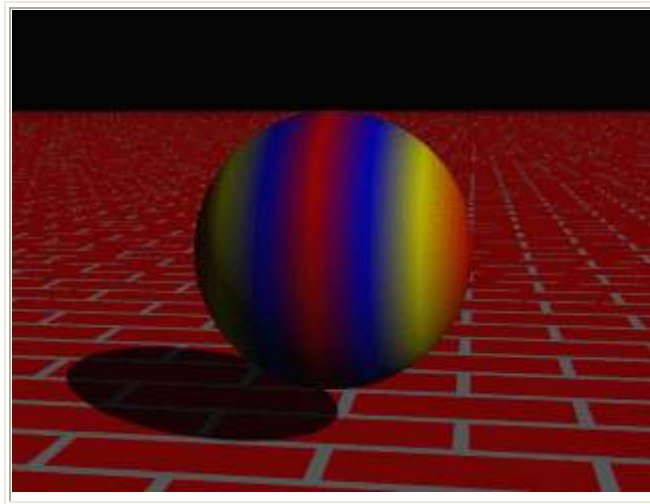



Fig. 95-Gradiente lungo l'asse x

vediamo un interessante sequenza di strisce verticali (gradiente). Cambiamo la direzione del gradiente a y.

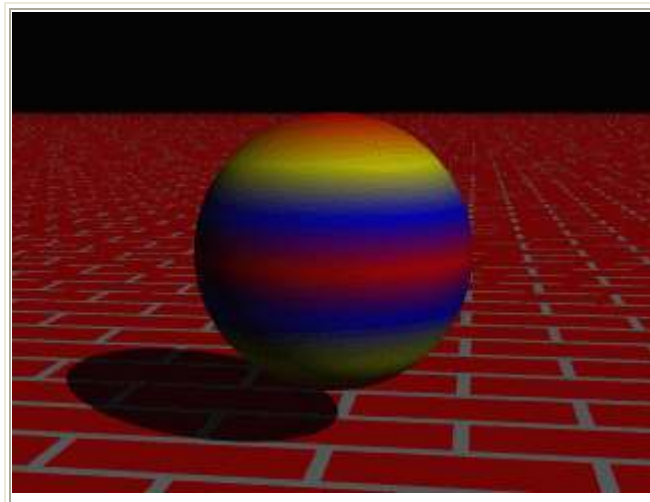



Fig. 96-Gradiente lungo l'asse delle y

Le strisce sono diventate orizzontali. Cambiamo di nuovo la direzione del gradiente a z.

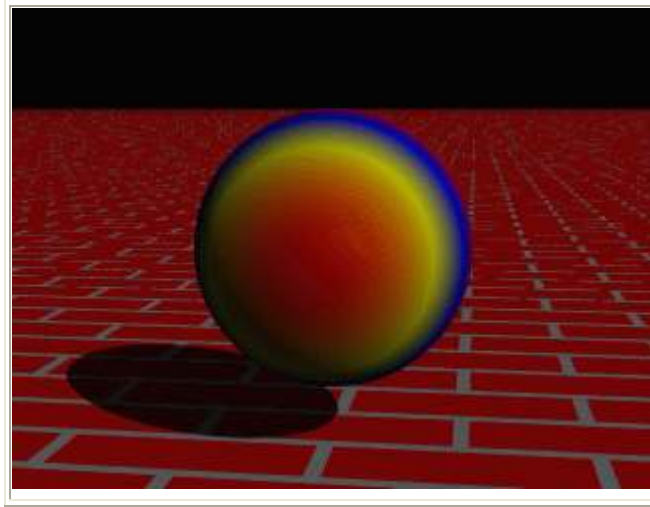



Fig. 97-Gradiente lungo l'asse dele z

Le strisce sono come anelli concentrici. Questo avviene perché la direzione del gradiente punta dalla parte opposta alla camera. Rimettiamo la direzione del gradiente ad x e modifichiamo la frase `pigment` nel seguente modo :

```
pigment {
gradient x
color_map {
[0.00 color Red]
[0.33 color Blue]
[0.66 color Yellow]
[1.00 color Red]
}
rotate -45*z // abbiamo aggiunto questa linea
}
```

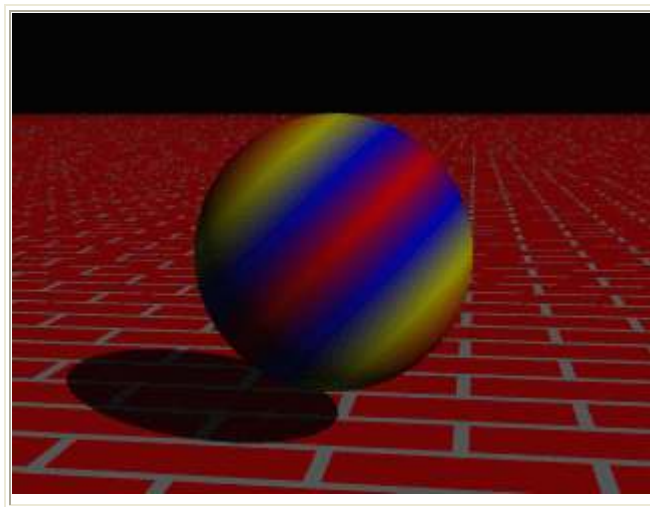



Fig. 98-Gradiente ruotato.

Le bande verticali sono ora inclinate di un angolo di 45°. Tutti i motivi possono essere ruotati, ridimensionati e traslati in questo modo. Proviamo ora pattern di tipo diverso. Uno per volta,

sostituiamo a `gradient x` le seguenti parole chiave e renderizziamo per vedere le differenze :
`bozo`,

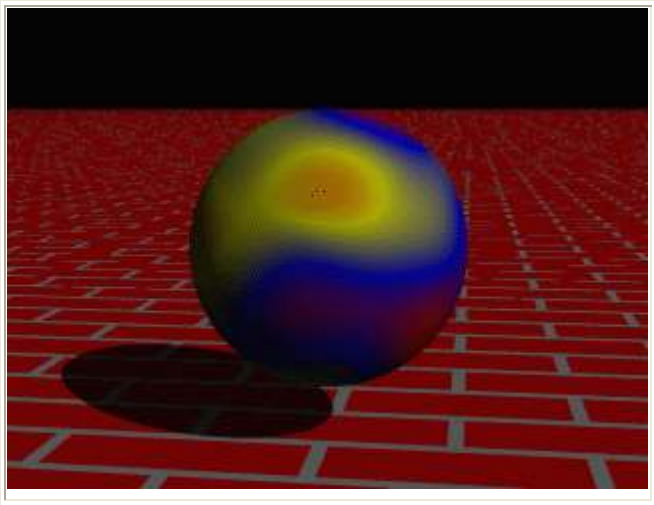



Fig. 99-Bozo

`marble`,

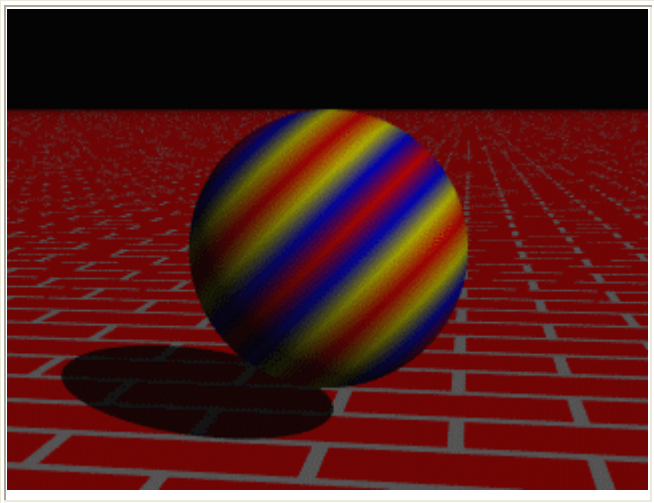



Fig.100-Marble

`agate`,

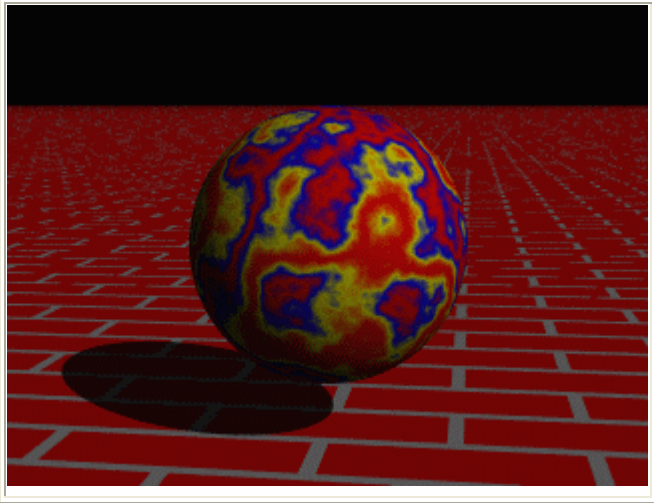



Fig.101-Agate

`granite,`

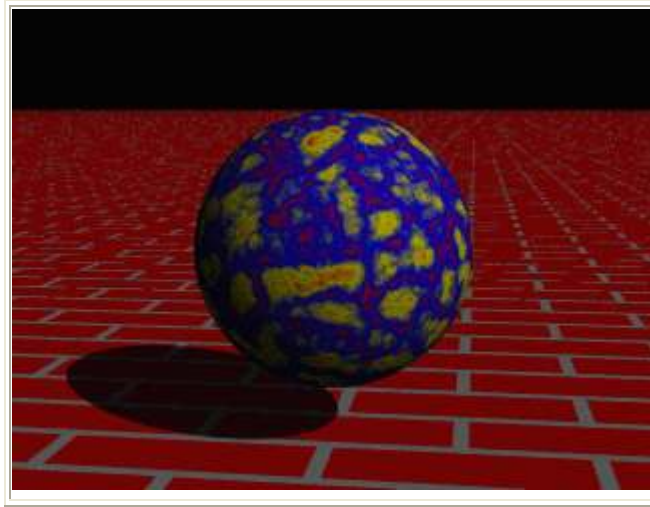



Fig. 102-Granite

`leopard,`

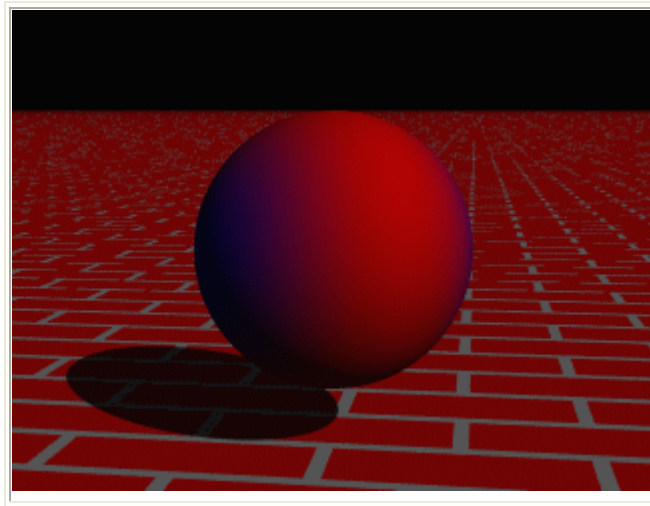



Fig. 103-Leopard

`spotted`

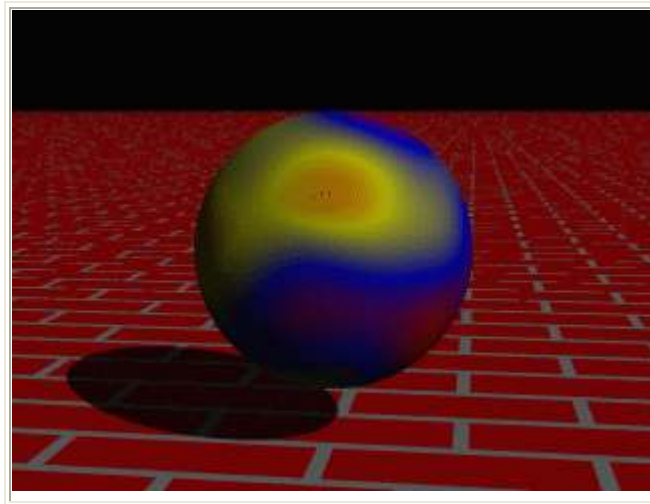



Fig. 104-Spotted

e `wood`

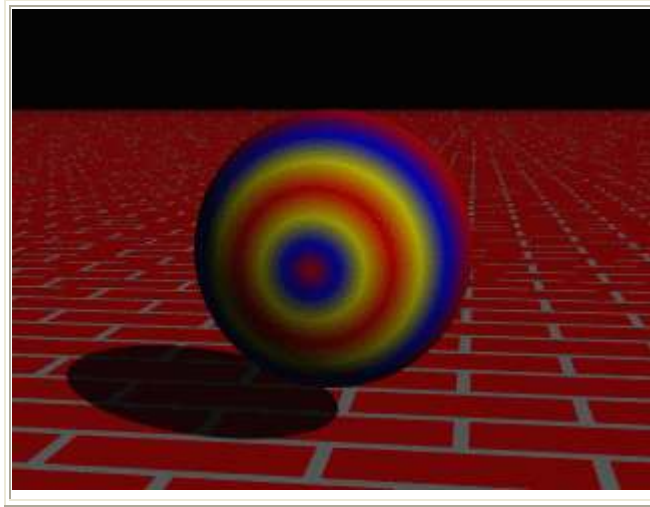



Fig.105-Wood

Renderizzando questi campioni, vediamo che ciascuno dà un motivo leggermente diverso dagli altri. Ma per avere risultati veramente buoni, ogni pattern ha bisogno di qualche modificatore di pattern.

4.8.2.3 Usare Modificatori di Pattern

Vediamo qualche modificatore di pattern. Per prima cosa, cambiamo il tipo di pattern a `bozo`. Poi, aggiungiamo il seguente cambiamento :

```
pigment {
bozo
frequency 3 // <- aggiungiamo questa linea
color_map {
[0.00 color Red]
[0.33 color Blue]
[0.66 color Yellow]
[1.00 color Red]
}
rotate -45*z
}
```

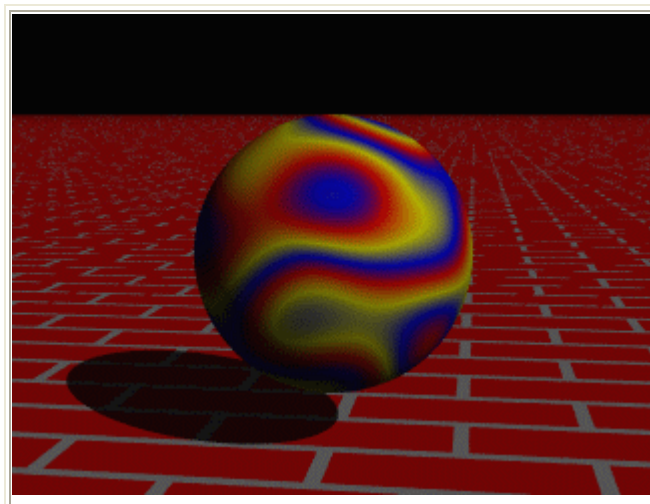



Fig. 106-Aggiungere la frequenza

Il modificatore `frequency` (frequenza) determina il numero di volte che la mappa di colori si ripete per unità di dimensione. Questo cambiamento causa la presenza di un numero maggiore di bande di colore, nel nostro pigmento `bozo`, di quante ne avevamo prima. Ora, sostituiamo `bozo` con `marble`.

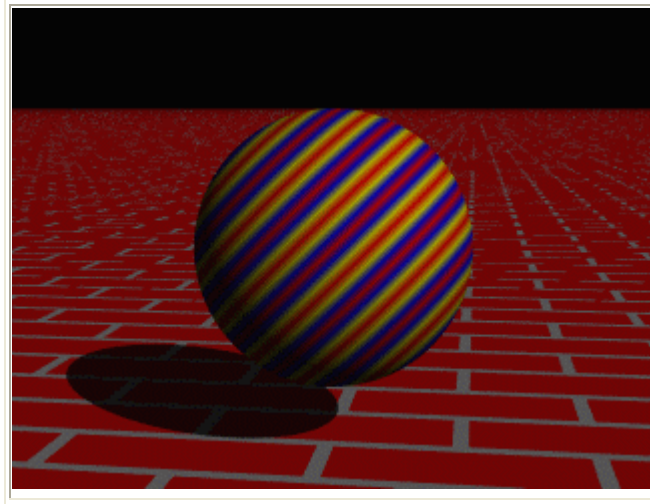



Fig. 107-Marble

Quando lo avevamo renderizzato prima, avevamo visto un motivo a strisce, simile a `gradient y`, ma per nulla somigliante a del marmo. Questo perché il marmo è in effetti una specie di gradiente di colore, ma ha bisogno di un altro modificatore per essere realistico. Questo modificatore si chiama `turbulence` (turbolenza). Sostituiamo `frequency 3` con `turbulence 1` e renderizziamo di nuovo.

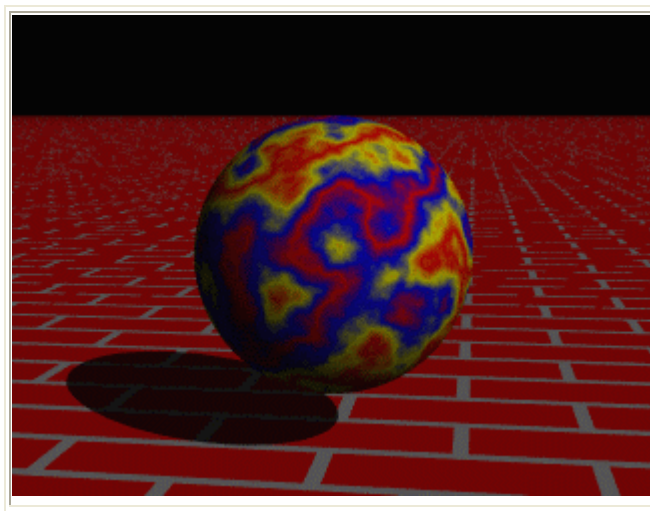



Fig. 108-Aggiungere turbolenza

Ora va meglio ! Ora rimettiamo `frequency 3` alla riga dopo `turbulence 1` e diamo un'altra occhiata. Ancora più interessante !

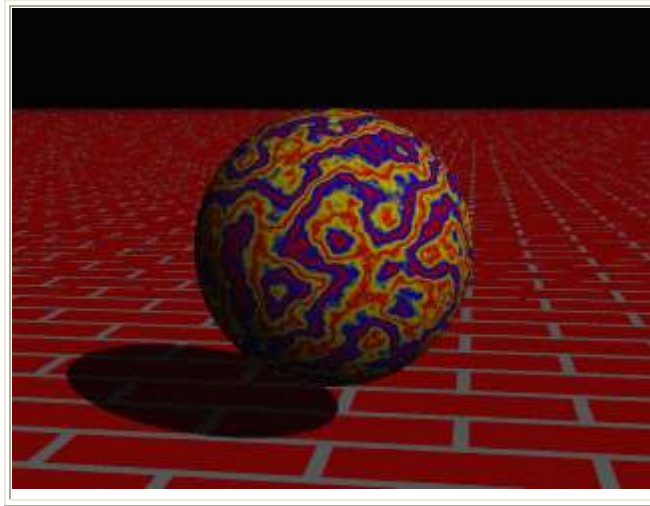



Fig. 109-Frequenza e turbolenza

Ma aspetta, può migliorare ! La stessa turbolenza possiede alcuni modificatori 'personali'. Possiamo infatti modificarla in alcuni modi. Primo, il numero decimale che segue la parola chiave `turbulence` può assumere qualunque valore, numeri più alti ci forniscono una maggiore turbolenza. Secondo, possiamo usare le parole chiave `omega`, `lambda` ed `octaves` per modificare i parametri della turbolenza. Proviamo :

```
pigment {
marble
turbulence 0.5
lambda 1.5
omega 0.8
octaves 5
frequency 3
color_map {
[0.00 color Red]
[0.33 color Blue]
[0.66 color Yellow]
[1.00 color Red]
}
rotate 45*z
}
```

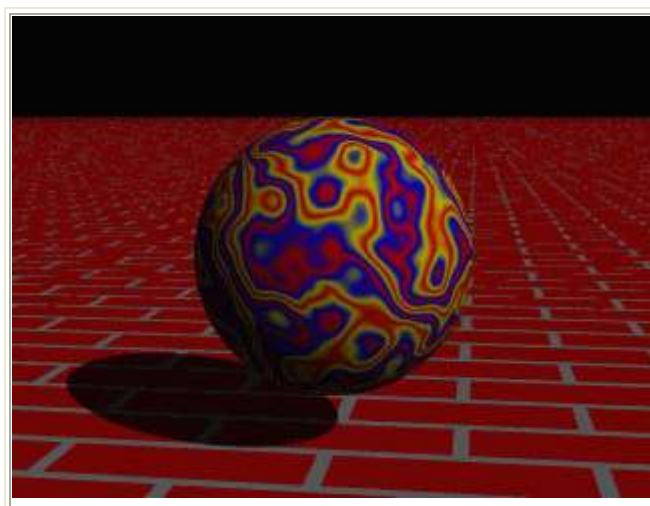



Fig. 110-Modificare gli altri parametri

Renderizzando questo esempio, vediamo che la turbolenza è cambiata ed il motivo ha un aspetto diverso. Divertiamoci per un po' con i valori di `turbulence`, `lambda`, `omega` e `octaves` per vedere cosa succede.

4.8.2.4 Usare Pigmenti Trasparenti e Texture Stratificate

I pigmenti sono descritti da variabili numeriche che ci forniscono i valori di rosso, verde e blu del colore da usare (per esempio, `color rgb <1, 0, 0>` ci dà il rosso 'puro'). Ma questa sintassi può darci più che i valori di rosso, verde e blu. Possiamo specificare la trasparenza modificandola come segue : `color rgbf<1, 0, 0, 1>`. La `f` sta per `filter` (filtro), la parola che POV-Ray usa per la trasparenza filtrata. Un valore di 1 significa che l'oggetto è completamente trasparente, ma che tuttavia filtra la luce in accordo al colore specificato. In questo caso, avremmo un rosso trasparente, come cellophane. Esiste un altro tipo di trasparenza in POV-Ray. Si chiama trasmittanza o trasparenza non filtrata (la parola chiave è `transmit`). E' diversa dalla trasparenza filtrata nel fatto che non modifica il colore della luce che passa attraverso. Può essere specificata in questo modo : `rgbt <1, 0, 0, 1>`.
Utilizziamo alcuni pigmenti trasparenti per creare un altro tipo di texture, la texture stratificata. Ritornando al nostro precedente esempio, dichiariamo la seguente texture :

```
#declare LandArea = texture {
pigment {
agate
turbulence 1
lambda 1.5
omega .8
octaves 8
color_map {
[0.00 color rgb <.5, .25, .15>]
[0.33 color rgb <.1, .5, .4>]
[0.86 color rgb <.6, .3, .1>]
[1.00 color rgb <.5, .25, .15>]
}
}
}
}
```

Questa texture costituirà la terraferma. Ora, facciamo gli oceani dichiarando la seguente :

```
#declare OceanArea = texture {
pigment {
bozo
turbulence .5
lambda 2
color_map {
[0.00, 0.33 color rgb <0, 0, 1>
color rgb <0, 0, 1>]
[0.33, 0.66 color rgbf <1, 1, 1, 1>
color rgbf <1, 1, 1, 1>]
[0.66, 1.00 color rgb <0, 0, 1>
color rgb <0, 0, 1>]
}
```

```
}
}
}
```

Nota come l'oceano è la zona blu opaca e la terraferma abbia delle zone trasparenti che permettono di vedere la texture sottostante. Ora, dichiariamo un'altra texture per simulare un atmosfera con nubi frastagliate :

```
#declare CloudArea = texture {
pigment {
agate
turbulence 1
lambda 2
frequency 2
color_map {
[0.0 color rgbf <1, 1, 1, 1>]
[0.5 color rgbf <1, 1, 1, .35>]
[1.0 color rgbf <1, 1, 1, 1>]
}
}
}
```

Ora, applichiamo tutte e tre queste texture alla nostra sfera.

```
sphere { <0,0,0>, 1
texture { LandArea }
texture { OceanArea }
texture { CloudArea }
}
```

Renderizziamo questo esempio ed otteniamo un pianetino abbastanza fedele.

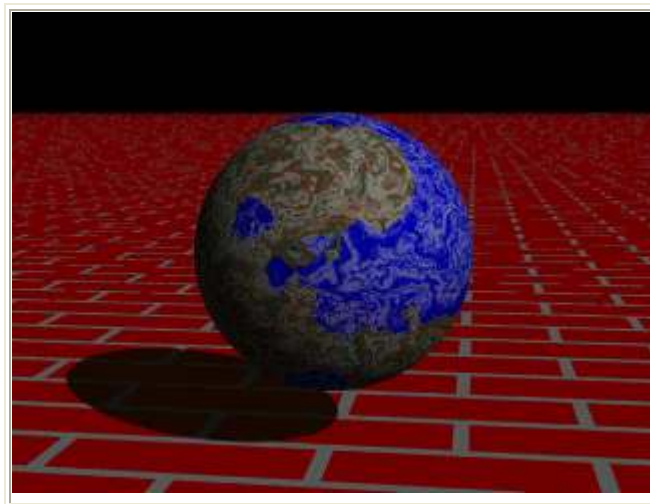



Fig.111-Pianeta

Potrebbe essere meglio. Non ci piace, in particolare, l'aspetto delle nubi. C'è un modo in cui potrebbero essere fatte, che sarebbe molto più realistico.

4.8.2.5 Usare Mappe di Pigmentazione.

I pigmenti possono essere fusi assieme nello stesso modo in cui si fondono i colori in una mappa di colori usando le stesse parole chiave che si usano per i pigmenti. Proviamo. Aggiungiamo le seguenti dichiarazioni, assicurandoci che compaiano prima delle altre nel file.

```
#declare Clouds1 = pigment {
bozo
turbulence 1
color_map {
[0.0 color White filter 1]
[0.5 color White]
[1.0 color White filter 1]
}
}
#declare Clouds2 = pigment {
agate
turbulence 1
color_map {
[0.0 color White filter 1]
[0.5 color White]
[1.0 color White filter 1]
}
}
#declare Clouds3 = pigment {
marble
turbulence 1
color_map {
[0.0 color White filter 1]
[0.5 color White]
[1.0 color White filter 1]
}
}
#declare Clouds4 = pigment {
granite
turbulence 1
color_map {
[0.0 color White filter 1]
[0.5 color White]
[1.0 color White filter 1]
}
}
```

Ed ora, usiamo questi pigmenti che abbiamo appena dichiarato nel nostro strato di nuvole del pianetino. Sostituiamo il vecchio strato di texture delle nuvole con :

```
#declare CloudArea = texture {
pigment {
gradient y
pigment_map {
[0.00 Clouds1]
[0.25 Clouds2]
[0.50 Clouds3]
[0.75 Clouds4]
```

```
[1.00 Clouds1]
}
}
}
```

Renderizziamo questo esempio

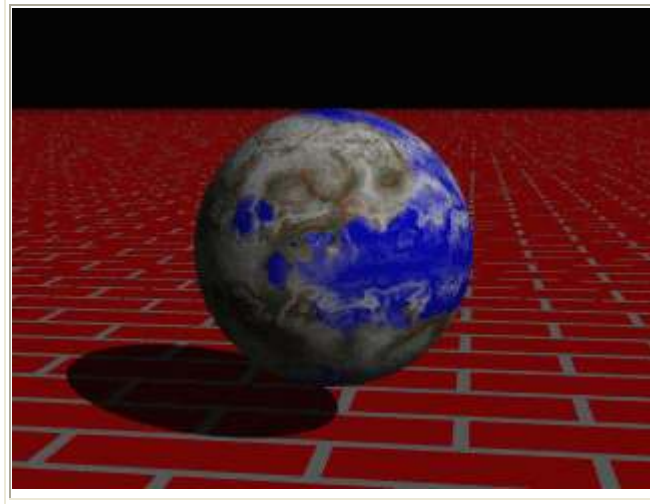



Fig.112-Pianeta

vediamo una disposizione delle nuvole che assomiglia molto a quella della Terra. Le nuvole sono divise in fasce, a simulare i diversi climi che si trovano a diverse latitudini.

4.8.3 Normali

Gli oggetti, in POV-Ray hanno superfici molto lisce. Ciò non è molto realistico e quindi abbiamo vari modi per disturbare la regolarità della superficie di un oggetto, perturbando la normale alla superficie. La normale alla superficie è il vettore che si trova perpendicolare alla superficie. Modificando questo vettore, la superficie può essere resa irregolare, grinzosa, od ancora in uno dei molti motivi disponibili. Proviamone un paio.

4.8.3.1 Modificatori di Normali

Commentiamo il nostro pianetino, per ora e in fondo al file creiamo una nuova sfera con un pigmento semplice, ad un colore solo.

```
sphere { <0,0,0>, 1
pigment { Gray75 }
normal { bumps 1 scale .2 }
}
```

Qui abbiamo aggiunto un blocco `normal` in aggiunta al blocco di istruzioni `pigment`. Nota che non c'è bisogno di racchiuderli in una frase `texture{...}`a meno che non debbano essere trasformati insieme, o andare a far parte di una texture stratificata. Renderizziamo questo esempio per vedere cosa ne esce.

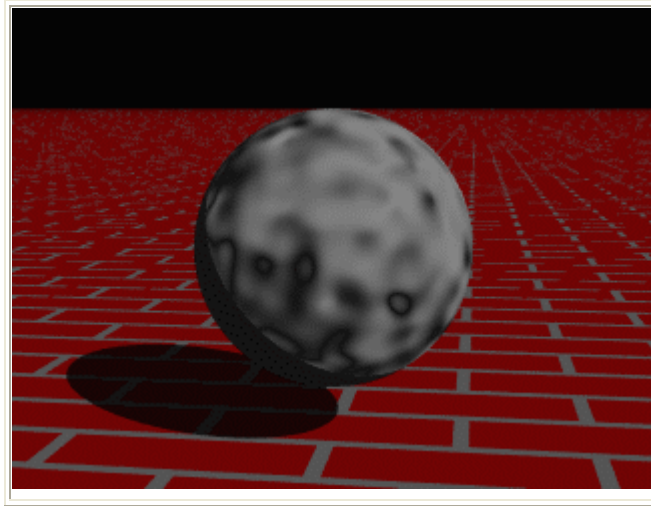



Fig. 113-Bumps

Ora, uno alla volta, sostituiamo a `bumps` le seguenti parole chiave : `dents`,

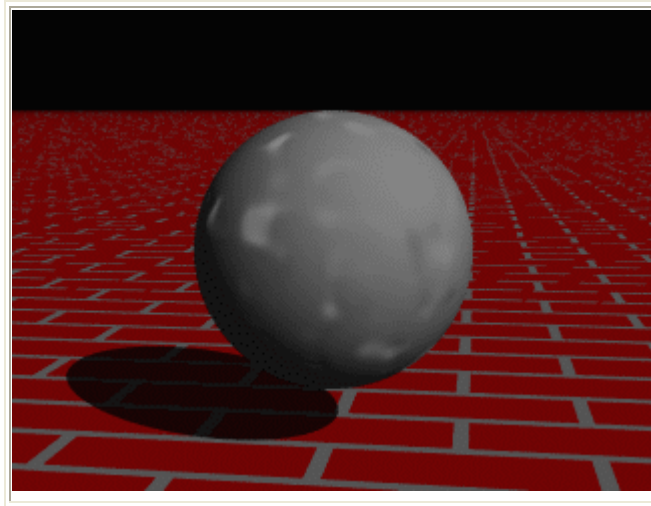



Fig. 114-Dents

`wrinkles`,

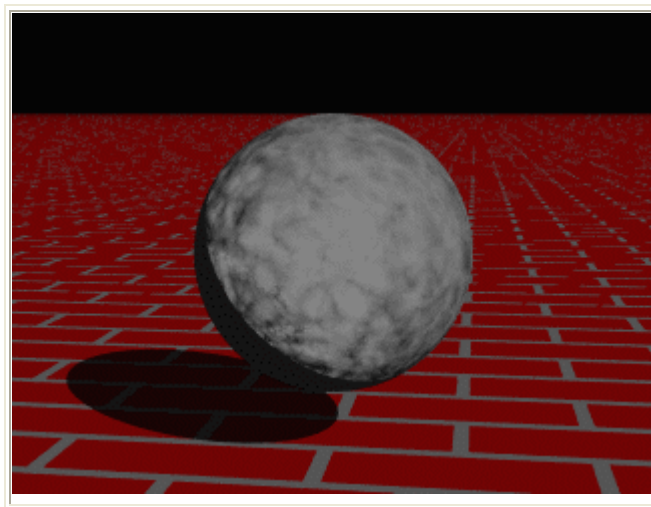



Fig. 115-Wrinkles

`ripples`

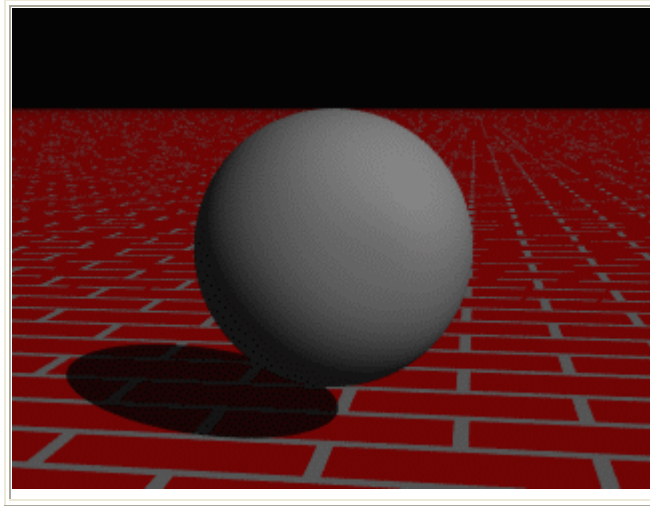



Fig. 116-Ripples

e `waves`

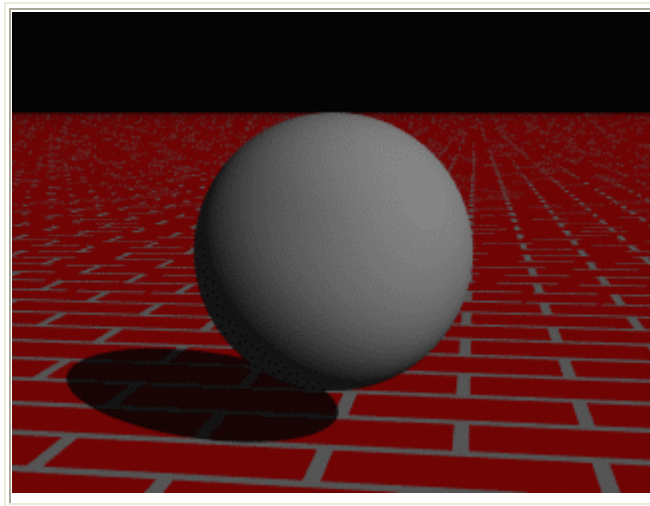



Fig. 117-Waves

(possiamo anche usare tutti i pattern elencati in "Pattern". Renderizziamo ognuno dei campioni per vedere cosa si ottiene. Sperimentiamo diversi valori del numero decimale che segue la parola chiave. Proviamo anche a ridiemnsionare il blocco `normal{...}` con l'istruzione `scale`. Per maggiore interesse, cambiamo la texture del piano in una texture con normali come segue :

```
plane { y, -1.5
pigment { color rgb <.65, .45, .35> }
normal { dents .75 scale .25 }
}
```

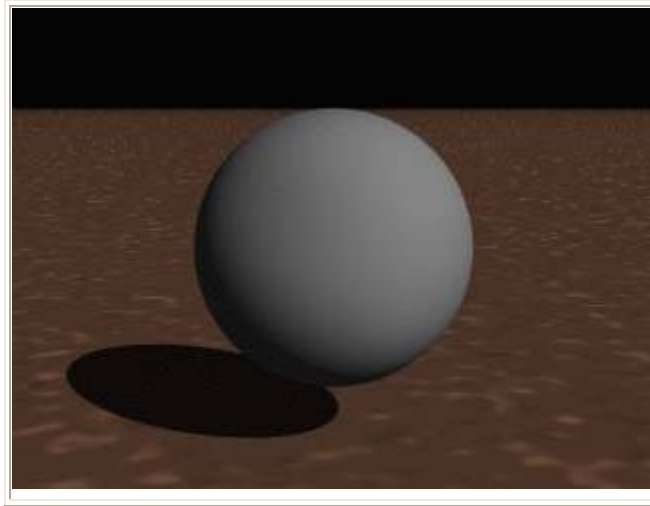



Fig. 118-Anche per il piano

4.8.3.2 Fondere le Normali

Le normali possono essere stratificate in maniera analoga ai pigmenti, ma i risultati possono essere inattesi. Proviamo, modificando la sfera come segue :

```
sphere { <0,0,0>, 1
pigment { Gray75 }
normal { radial frequency 10 }
normal { gradient y scale .2 }
}
```

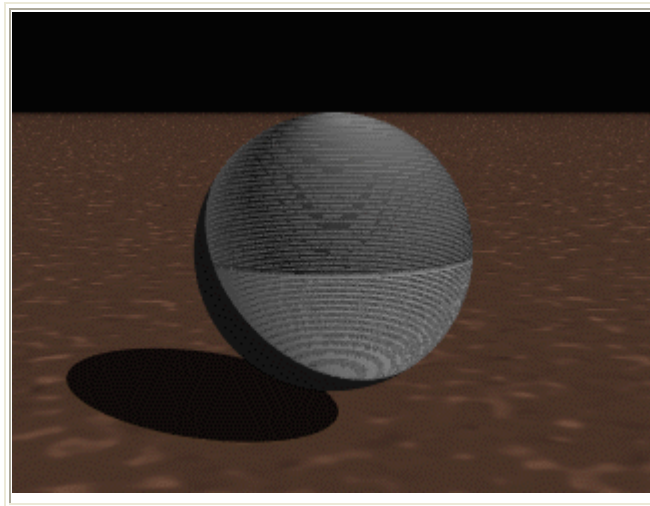



Fig. 118-Normali sovrapposte

Come possiamo vedere, il motivo risultante nelle normali non è né radiale, né un gradiente. E' invece il risultato che si ottiene prima calcolando un motivo radiale e poi sovrapponendogli additivamente un gradiente. Questo metodo può essere difficoltoso da controllare e così POV-Ray fornisce all'utente altri modi per unire insieme normali diverse.
Un metodo possibile è usare una mappa delle normali. Una mappa di normali funziona nello stesso modo in cui funzionano le mappe di pigmenti che abbiamo usato prima. Modifichiamo la texture della nostra sfera come segue :

```
sphere { <0,0,0>, 1
pigment { Gray75 }
```

```
normal {
gradient y
frequency 3
turbulence .5
normal_map {
[0.00 granite]
[0.25 spotted turbulence .35]
[0.50 marble turbulence .5]
[0.75 bozo turbulence .25]
[1.00 granite]
}
}
}
```

Renderizzando questo esempio

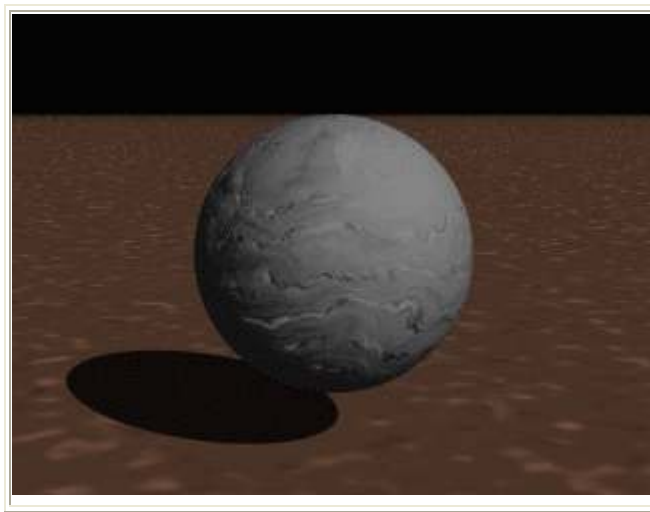



Fig. 119-Mappa di normali

Vediamo che la sfera ha ora una superficie molto irregolare. La disposizione delle normali lungo un pattern a gradiente le separa in fasce, ma queste sono irregolari, dando alla superficie un aspetto caotico. Ma questo ci da un'idea. Supponiamo di usare lo stesso motivo che abbiamo usato per creare gli oceani del nostro pianetino, per creare una mappa di normali e di applicarla alle zone emerse. Ne segue che se usassimo gli stessi motivi e gli stessi modificatori su una sfera della stessa dimensione, la forma del motivo sarebbe la stessa ? Renderebbe le zone di terraferma irregolari lasciando gli oceani lisci ? Proviamo. Per prima cosa, renderizziamo le due sfere una di fianco all'altra in modo da poter vedere se il motivo è veramente lo stesso. Leviamo il commento dal pianetino e lo modifichiamo nel seguente modo :

```
sphere { <0,0,0>, 1
texture { LandArea }
texture { OceanArea }
//texture { CloudArea } // <-commenta questa riga
translate -x // <- aggiungi questa trasformazione
}
```

Ed ora, cambiamo la sfera grigia come segue :

```
sphere { <0,0,0>, 1
pigment { Gray75 }
```

```
normal {
bozo
turbulence .5
lambda 2
normal_map {
[0.4 dents .15 scale .01]
[0.6 agate turbulence 1]
[1.0 dents .15 scale .01]
}
}
translate x // <- aggiungi questa trasformazione
}
```

Renderizziamo per vedere se il motivo è lo stesso.

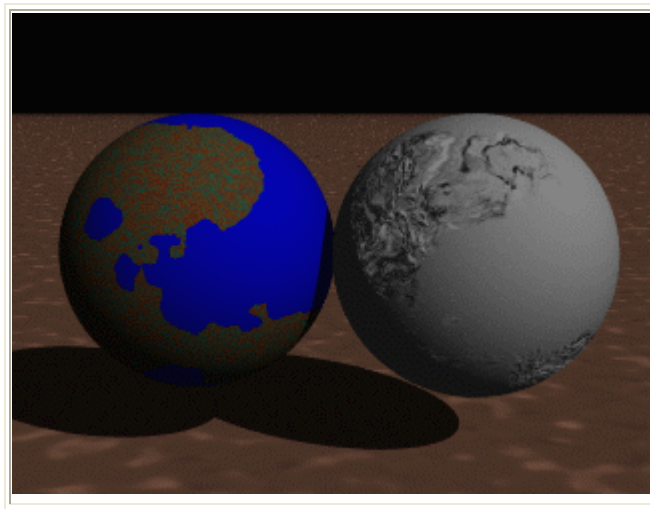



Fig.120-Le due sfere affiancate

Vediamo che sono uguali. Allora, commentiamo la sfera grigia ed aggiungiamo il blocco di normali che contiene alla texture che abbiamo assegnato alla terraferma del nostro pianetino. Eliminiamo le trasformazioni in modo da centrare nuovamente nella scena il pianeta.

```
#declare LandArea = texture {
pigment {
agate
turbulence 1
lambda 1.5
omega .8
octaves 8
color_map {
[0.00 color rgb <.5, .25, .15>]
[0.33 color rgb <.1, .5, .4>]
[0.86 color rgb <.6, .3, .1>]
[1.00 color rgb <.5, .25, .15>]
}
}
normal {
bozo
turbulence .5
lambda 2
```

```
normal_map {
[0.4 dents .15 scale .01]
[0.6 agate turbulence 1]
[1.0 dents .15 scale .01]
}
}
}
```

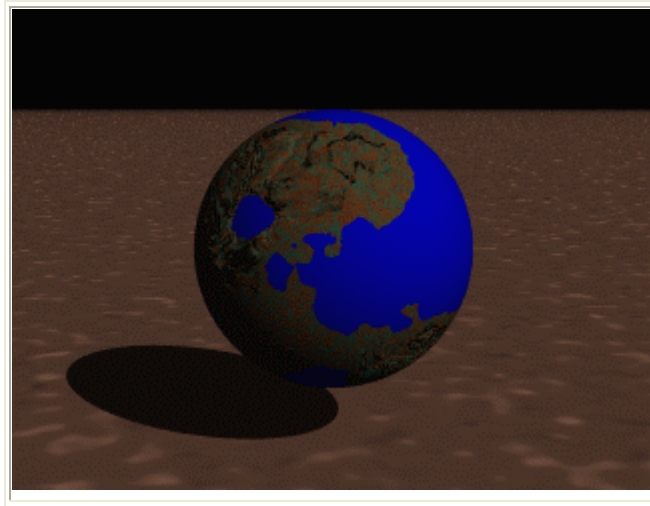



Fig.121-Abbiamo applicato le normali alla terraferma

Guardando l'immagine risultante, vediamo che la nostra idea funziona ! Le zone di terraferma sono irregolari, mentre gli oceani sono piatti. Rimettiamo a posto lo strato di nuvole ed il nostro pianetino è completo.

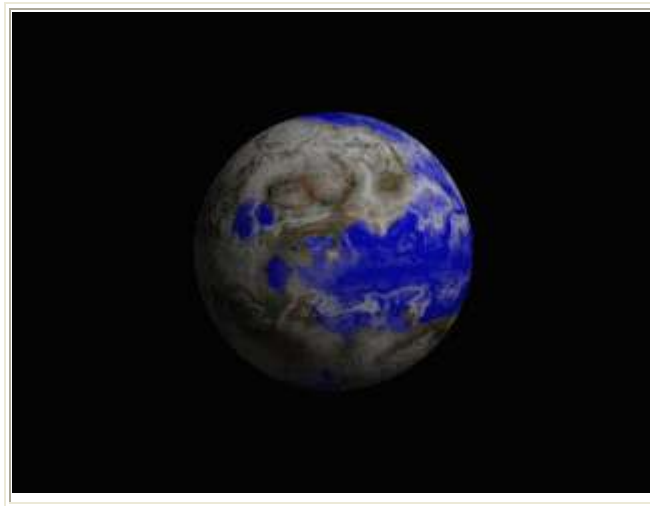



Fig. 122-Il pianeta completo

C'è molto di più di quanto non abbiamo esposto qui per motivi di spazio. Da soli, dovremmo prenderci il tempo di esplorare mappe slope (vedi § 7.6.2.1), mappe bump (vedi § 7.6.2.3) e `average` (vedi § 7.6.7.2).

4.8.4 Finitura

La parte finale di una texture in POV-Ray è la finitura. Controlla le proprietà della superficie di un oggetto. Può renderlo brillante e riflettente, oppure opaco e piatto. Può anche specificare cosa succede alla luce che passa attraverso pigmenti trasparenti, cosa succede alla luce che è dispersa da una superficie non perfettamente liscia e cosa succede alla luce che viene riflessa da superfici con

proprietà di interferenza da strato sottile. Ci sono 12 differenti proprietà disponibili in POV-Ray per specificare le finitura di un oggetto. Sono controllate dalle seguenti parole chiave : `ambient`, `diffuse`, `brilliance`, `phong`, `specular`, `metallic`, `reflection`, `refraction`, `caustics`, `attenuation`, `crand` e `iridescence`. Inventiamo un paio di texture per usare questi parametri.

4.8.4.1 Luce Ambiente

Dato che gli oggetti in POV-Ray sono illuminati dalle sorgenti luminose, le loro parti che si trovano in ombra sarebbero totalmente nere, se non fosse per le prime due proprietà della finitura, `ambient` e `diffuse`. La luce ambiente (`ambient`) è utilizzata per simulare la luce, dispersa nella scena, che non proviene direttamente da una sorgente luminosa. La luce diffusa (`diffuse`) determina quanta della luce che si vede proviene direttamente dalle sorgenti luminose. Queste due parole chiave, insieme, controllano la simulazione della luce ambiente. Useremo la nostra sfera grigia per dimostrare questo. Modifichiamo anche il nostro piano, riportandolo all'origniale scacchiera bianca e verde.

```
plane {y,-1.5
pigment {checker Green, White}
}

sphere { <0,0,0>, 1
pigment {Gray75}
finish {
ambient .2
diffuse .6
}
```

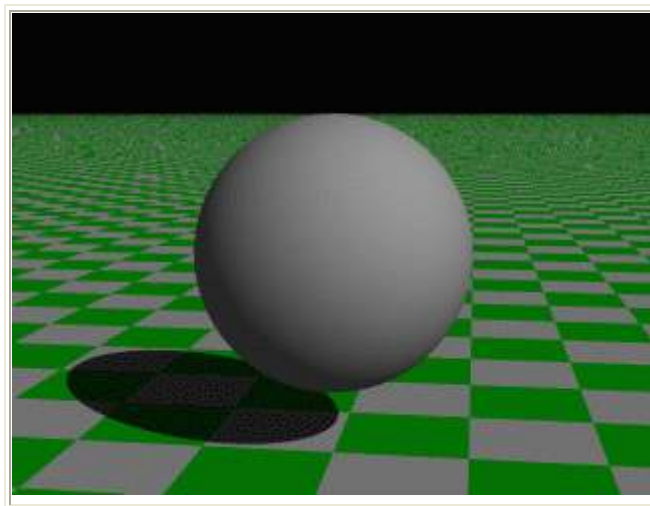



Fig. 123-I valori predefiniti per la luce ambiente equella diffusa

Nell'esempio sopra, usiamo i valori predefiniti per la luce ambiente e quella diffusa. Renderizziamo questa scena per vedere l'effetto e quindi, modifichiamo la frase `finish` nel modo seguente :

```
ambient 0
diffuse 0
```

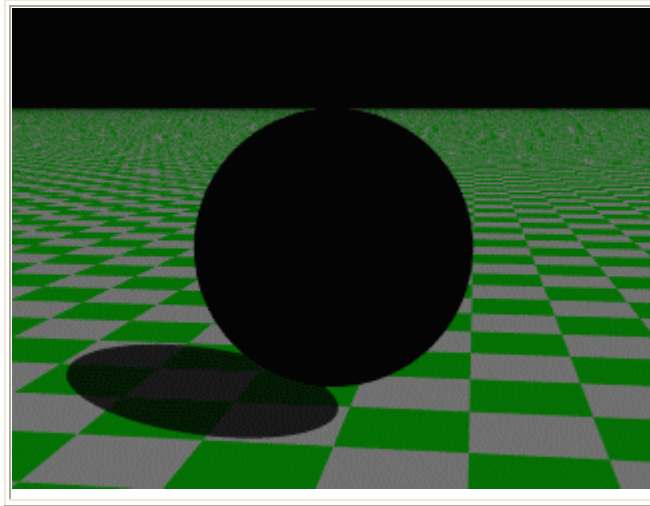



Fig. 124-Senza luce ambiente e diffusa

La sfera è nera, perché abbiamo specificato che nessuna luce, proveniente da nessuna sorgente luminosa, venga riflessa dalla sfera. Riportiamo il valore di `diffuse` a 0.6.

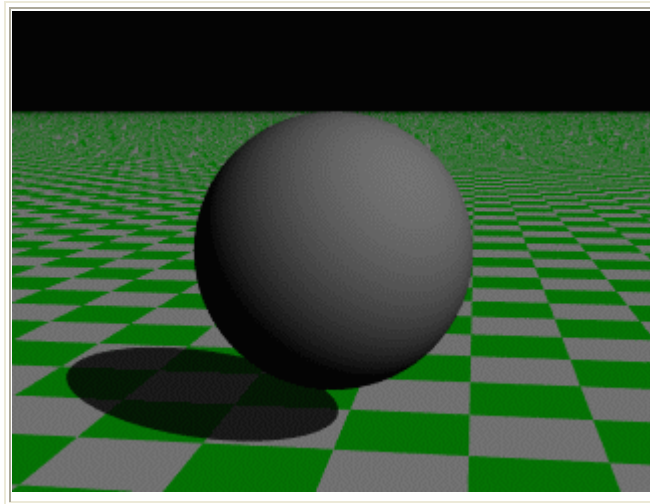



Fig. 125-Solo luce diffusa

Ora vediamo la superficie grigia dove la luce proveniente dalla sorgente luminosa illumina direttamente la sfera, ma la parte non direttamente illuminata è ancora completamente nera. Ora, cambiamo `diffuse` ed `ambient` a 0.3.

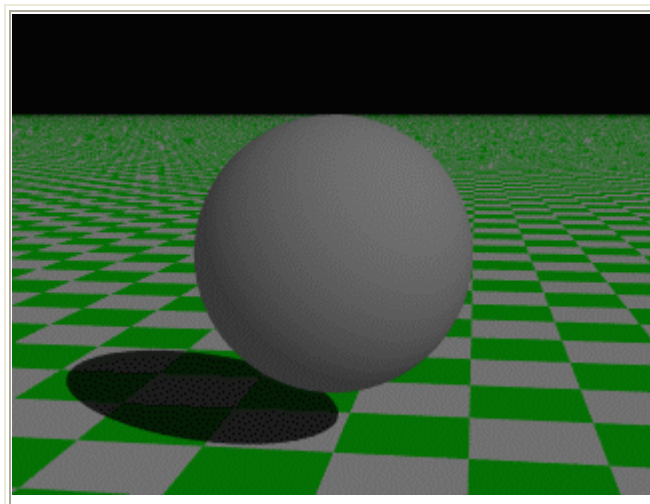



Fig. 126-Luce ambiente e luce diffusa al valore di 0.3

Ora, la sfera sembra quasi piatta. Questo avviene perché abbiamo specificato una quantità abbastanza alta di luce ambiente, mentre solo una piccola quantità della luce proveniente dalla sorgente luminosa è riflessa verso la camera. I valori predefiniti di `ambient` e `diffuse` sono in media abbastanza buoni. Nella maggior parte dei casi, un valore di luce ambiente di 0.1...0.2 è sufficiente ed un valore per la luce diffusa di 0.5...0.7 andrà altrettanto bene. Ci sono un paio di eccezioni. Se abbiamo una superficie completamente trasparente con valori molto alti di riflessione e/o rifrazione, valori bassi sia per `ambient` che per diffuse possono essere la scelta migliore. Questo è un esempio.

```
sphere { <0,0,0>, 1
pigment { White filter 1 }
finish {
ambient 0
diffuse 0
reflection .25
refraction 1
ior 1.33
specular 1
roughness .001
}
}
}
```

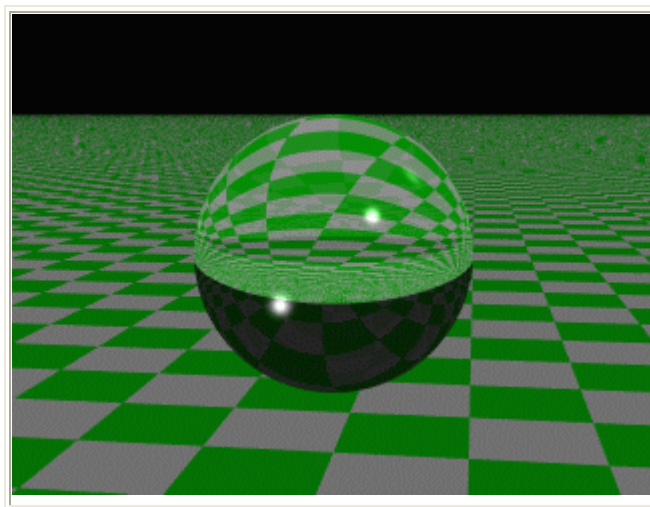



Fig. 127-Vetro

Questo è vetro, naturalmente. Il vetro è un materiale che acquista quasi tutto il suo aspetto da ciò che lo circonda. Si vede molto poco la superficie perché trasmette o riflette praticamente tutta la luce che riceve. (vedi il file **glass.inc** per altri esempi). Se abbiamo bisogno di un oggetto che sia completamente illuminato indipendentemente dalla situazione delle luci in una data scena, possiamo farlo artificialmente specificando un valore per `ambient` di 1 e di 0 per `diffuse`. Questo eliminerà tutte le ombre e fornirà, semplicemente, il più brillante valore di colore all'oggetto in tutti i suoi punti. Questo può essere utile per simulare oggetti che emettono luce come lampadine e per il cielo in scene dove esso non può essere adeguatamente illuminato con altri mezzi. Proviamo ora con la nostra sfera.

```
sphere { <0,0,0>, 1
pigment { White }
```

```
finish {
ambient 1
diffuse 0
}
}
}
```

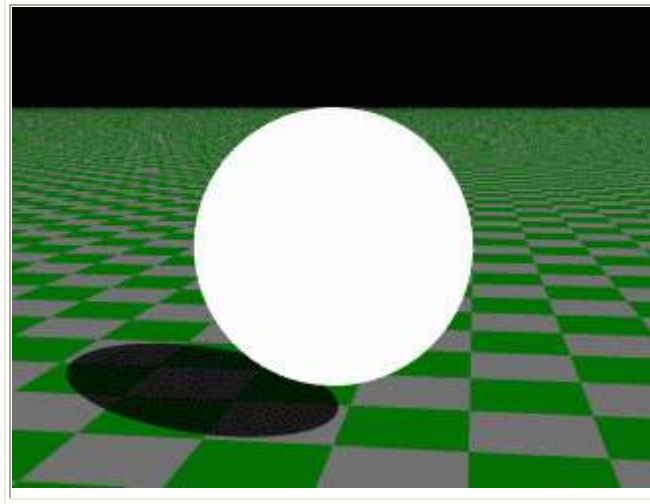



Fig. 127-Solo luce ambiente

Renderizzando questa scena otteniamo una sfera bianca, accecante, senza riflessi o parti in ombra. Potrebbe dare un buon lampione.

4.8.4.2 Punti di Riflessione

Nell'esempio del vetro che abbiamo fatto sopra, abbiamo notato dei piccoli punti luminosi sulla superficie della sfera. Questo dava alla sfera un aspetto duro e brillante. POV-Ray ci offre due modi di specificare punti di riflessione sulla superficie degli oggetti. Il primo si chiama riflessione di Phong. Normalmente, i riflessi di Phong sono descritti in POV-Ray usando due parole chiave : `phong` e `phong_size`. Il numero decimale che segue `phong` determina quanto illuminato deve essere il riflesso, mentre il numero decimale che segue `phong_size` determina la sua dimensione. Proviamo questo :

```
sphere { <0,0,0>, 1
pigment { Gray50 }
finish {
ambient .2
diffuse .6
phong .75
phong_size 25
}
}
```

Renderizzando questo esempio

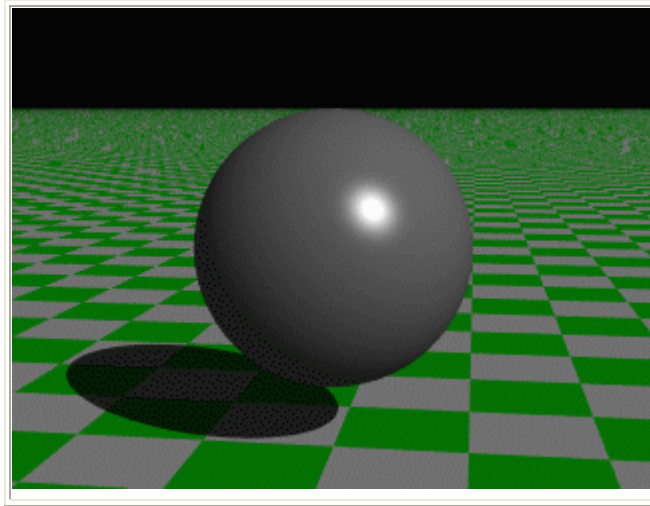



Fig. 123-Aspetto plastico

vediamo un riflesso piuttosto largo, sfumato, che dà alla sfera un aspetto simile alla plastica. Ora cambiamo `phong_size` a 150.

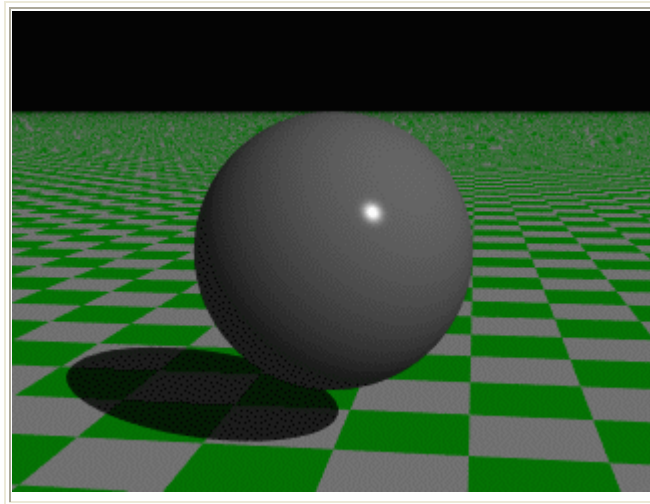



Fig. 124-Più brillante

Questo ci dà un punto di riflessione molto più piccolo che fa sembrare la sfera di un materiale molto più duro e brillante.
Esiste un secondo metodo di calcolare i riflessi, chiamato riflessione speculare. Viene specificato usando la parola chiave `specular` ed opera insieme ad un altra parola chiave, `roughness` (ruvidità). Queste due parole chiave lavorano insieme in un modo molto simile a quello in cui operano `phong` e `phong_size` per creare riflessi che modificano la lucentezza della superficie.
Proviamo ad usare `specular` sulla nostra sfera.

```
sphere { <0,0,0>, 1
pigment { Gray50 }
finish {
ambient .2
diffuse .6
specular .75
roughness .1
}
```

```
}
}
```

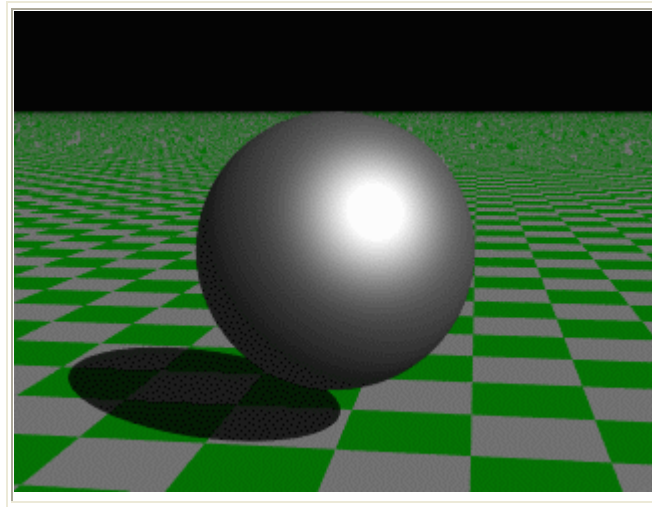



Fig. 125-Riflesso ampio

Guardando il risultato, vediamo un riflesso largo e sfumato, simile a quello che avevamo ottenuto usando `phong_size` uguale a 25. Cambiamo il valore di `roughness` a 0.001 e renderizziamo di nuovo.

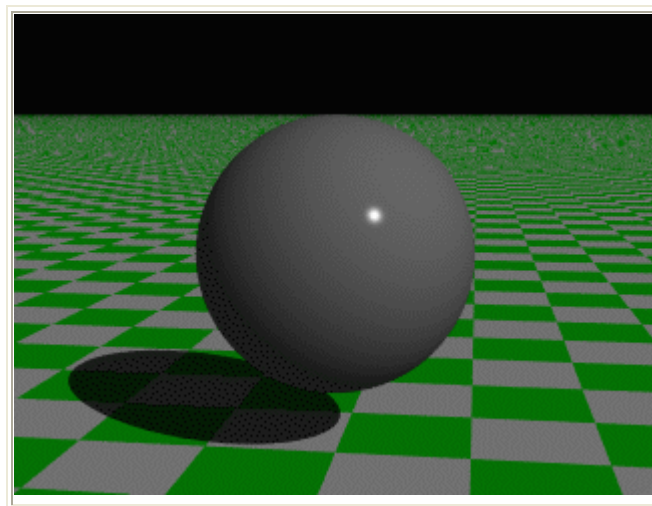



Fig.126-Riflesso più nitido

Ora vediamo un riflesso piccolo e netto, simile a quello che avevamo ottenuto con phong_size uguale a 150. In generale, la riflessione speculare è leggermente più accurata della riflessione di Phong, ma nel progettare una texture si dovrebbero provare entrambi i metodi. Esistono anche casi in cui entrambi i metodi possono essere utilizzati assieme in una finitura.

4.8.4.3 Utilizzo di Reflection e Metallic

C'è un altro parametro per simulare la finitura delle suoperfici, `reflection` (riflessione speculare). Superfici molto brillanti normalmente hanno un alto grado di riflessione. Vediamo un esempio :

```
sphere { <0,0,0>, 1
pigment { Gray50 }
finish {
ambient .2
```

```
diffuse .6
specular .75
roughness .001
reflection .5
}
}
}
```

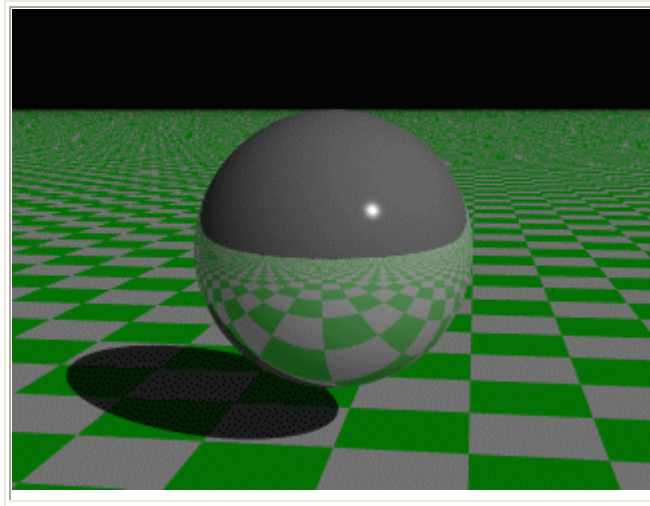



Fig. 126-Alto livello di riflessione speculare

Vediamo che la nostra sfera ora riflette il piano a scacchi bianchi e verdi e lo sfondo nero, ma il colore grigio della sfera sembra inadatto. Questo è un altro caso in cui abbiamo bisogno di un basso valore per la luce diffusa. In generale, maggiore è la riflessione, minore dovrebbe essere la luce diffusa. Abbassiamo il valore della luce diffusa a 0.3 e quello della luce ambiente a 0.1 e renderizziamo nuovamente.

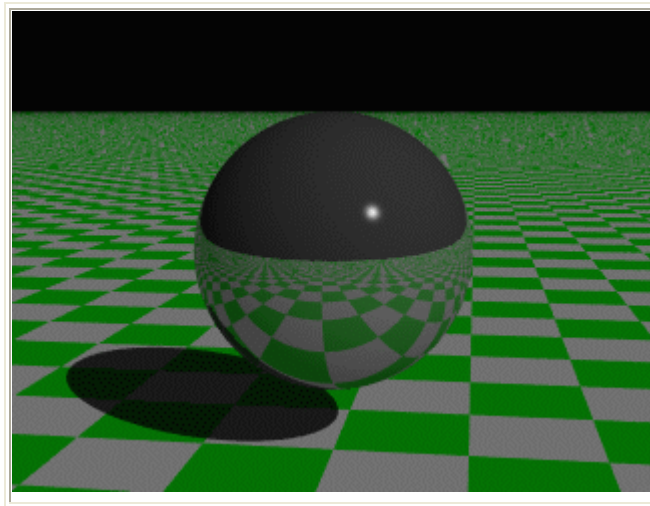



Fig. 127-Valori più bassi di luce ambiente e diffusa

Ora va molto meglio. Facciamo brillare la nostra sfera come una palla d'oro lucidata.

```
sphere { <0,0,0>, 1
pigment { BrightGold }
finish {
ambient .1
diffuse .1
```

```
specular 1
roughness .001
reflection .75
}
}
}
```

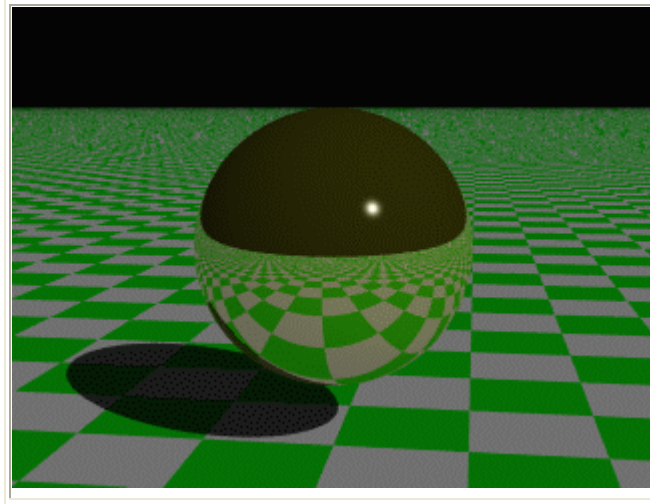



Fig. 128-Poca luce ambiente, poca luce diffusa e molta riflessione speculare

Ci siamo molto vicini, ma c'è qualcosa di sbagliato nel riflesso. Per rendere la superficie più simile al metallo, usiamo la parola chiave `metallic`. Aggiungiamola ora per vedere la differenza.

```
sphere { <0,0,0>, 1
pigment { BrightGold }
finish {
ambient .1
diffuse .1
specular 1
roughness .001
reflection .75
metallic
}
}
}
```

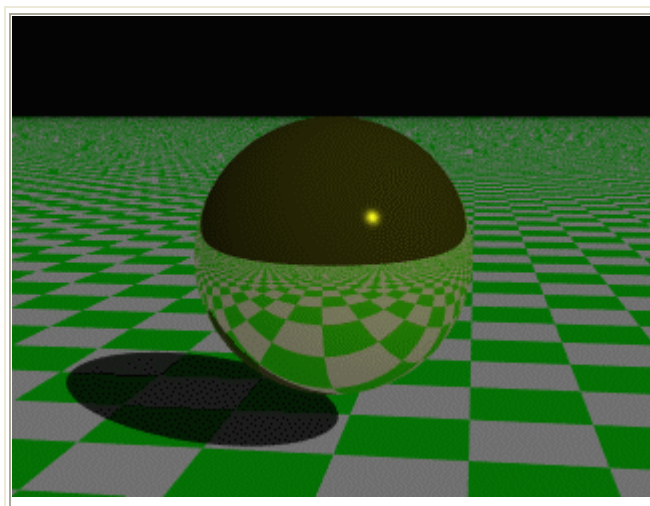



Fig. 129-La parola chiave metallic

Vediamo che il punto luminoso sulla sfera ha più il colore della superficie che quello della sorgente luminosa. Questo dà alla superficie un aspetto più metallico.

4.8.4.4 Rifrazione

Gli oggetti trasparenti permettono alla luce di attraversarli. Con alcuni materiali (in effetti, tutti tranne il vuoto assoluto, N.d.T.), i raggi di luce vengono curvati mentre viaggiano attraverso l'oggetto, a causa della diversa densità ottica degli oggetti. Questo fenomeno è chiamato *rifrazione*. L'acqua ed il vetro ad esempio 'piegano' la luce in questo modo. Per simulare adeguatamente l'acqua, o il vetro, POV-Ray ci fornisce un metodo per specificare la rifrazione, con le parole chiave `refraction` e `ior`. La quantità di luce che attraversa un oggetto è determinata dal valore di trasparenza filtrata e/o trasparenza non filtrata del pigmento. Dobbiamo usare il valore di `refraction` solo per attivare o disattivare la rifrazione della luce, usando valori di 1 o 0 rispettivamente (o gli argomenti booleani `on` e `off`). Vedi il paragrafo "Rifrazione" per una dettagliata spiegazione dei motivi. La quantità di rifrazione, cioè la quantità di deviazione della luce, è data dalla parola chiave `ior`, che sta per indice di rifrazione (*index of refraction*). Se conosciamo l'indice di rifrazione del materiale che stiamo creando, possiamo usarlo. Per esempio, l'acqua ha un indice di rifrazione di 1.33, il vetro di circa 1.45 ed il diamante di 1.75. Ritorniamo all'esempio della sfera di vetro che avevamo fatto prima.

```
sphere { <0,0,0>, 1
pigment { White filter 1 }
finish {
ambient 0
diffuse 0
reflection .25
refraction 1
ior 1.45
specular 1
roughness .001
}
}
}
```

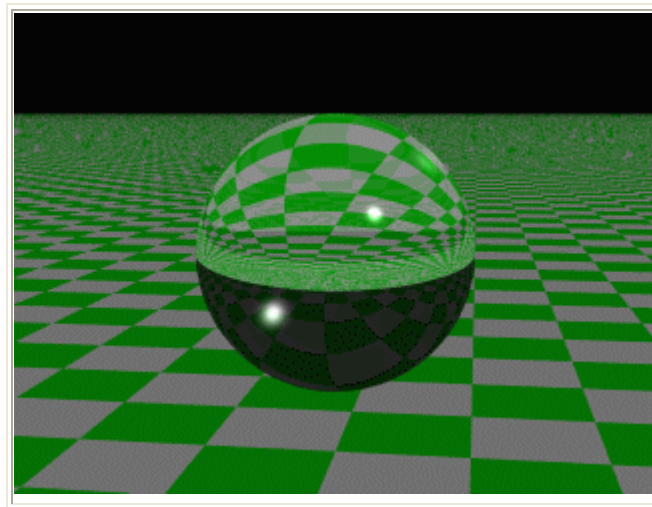



Fig. 130-Elevato indice di rifrazione

Renderizziamo nuovamente questo esempio e notiamo come il piano sia visibile attraverso la sfera distorto e ribaltato. Questo avviene perché la luce che passa attraverso la sfera è piegata (o rifratta)

della quantità specificata da `ior`. Riduciamo l'indice di rifrazione a 1.25 e renderizziamo nuovamente.

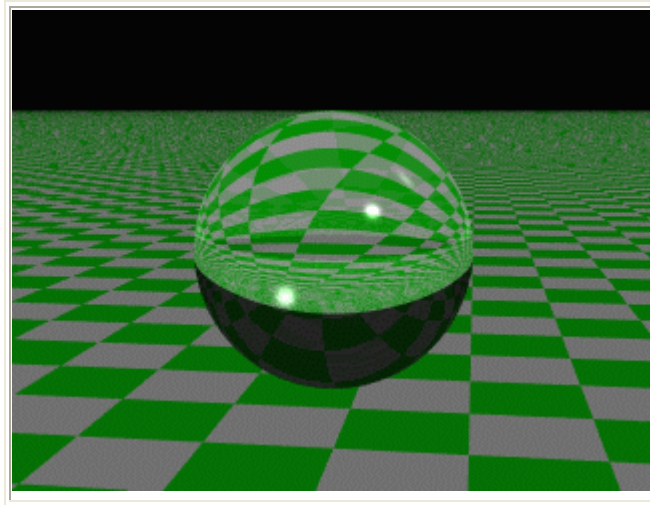



Fig. 131-Indice di rifrazione di 1.25

Aumentiamolo a 1.75 e renderizziamo di nuovo.

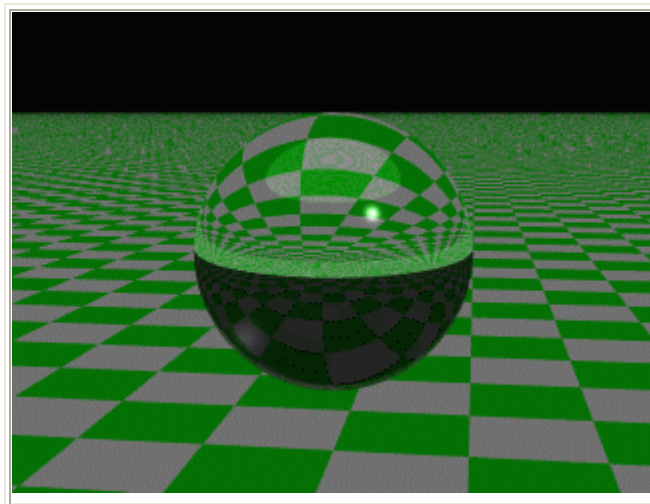



Fig. 132-Indice di rifrazione uguale a 1.75

Notiamo come cambia la distorsione.

4.8.4.5 Attenuazione delle Luci

Possiamo fare in modo che l'intensità della luce che attraversa un oggetto trasparente venga diminuita. Nella realtà questo è dovuto ad imperfezioni nella trasmissione della luce attraverso il mezzo. Due parametri determinano questo effetto : `fade_distance` è la distanza che la luce deve percorrere per raggiungere la metà della sua intensità, mentre `fade_power` rappresenta la velocità con cui diminuisce l'intensità. Proviamo un esempio :

```
sphere { <0,0,0>, 1
pigment { White filter 1 }
finish {
ambient .1
diffuse .1
reflection .15
refraction 1
```

```
ior 1.45
specular 1
roughness .001
fade_distance 5
fade_power 1
}
}
```

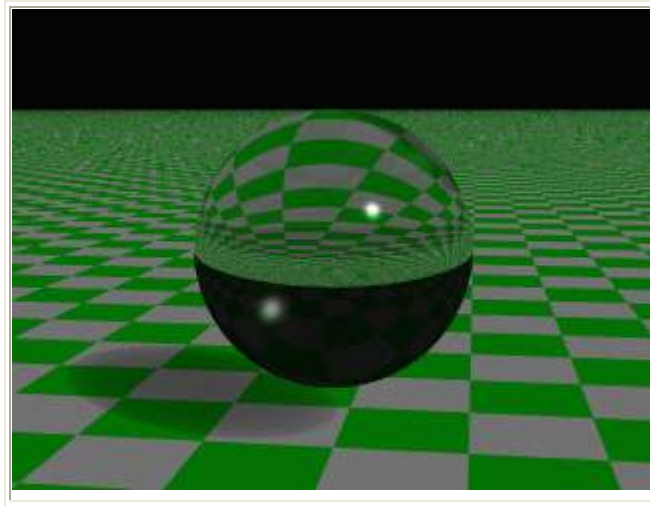



Fig. 133-Attenuare la luce che passa attraverso la sfera

Questo impartisce alla sfera un aspetto leggermente opaco, come se non tutta la luce potesse attraversarla. Per interessanti variazioni su questa texture, proviamo ad abbassare `ior` a 1.15 e ad alzare `reflection` a 0.5.

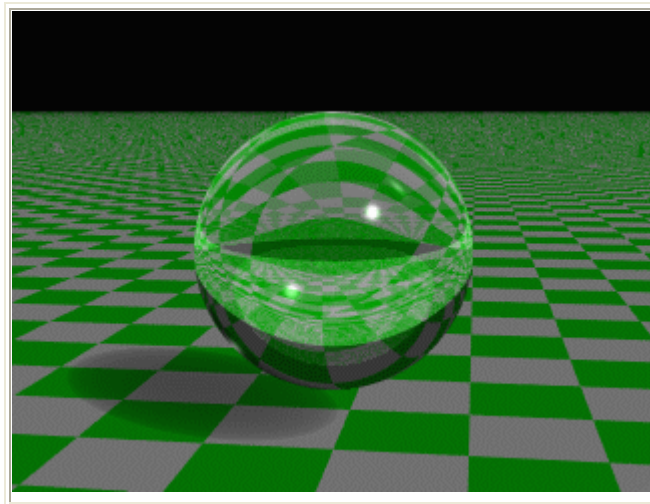



Fig. 135-Una variazione

4.8.4.6 Falsi Riflessi (Caustics)

4.8.4.6.1 Cosa Sono i Riflessi ?

Per prima cosa, saccheggiamo le mensole della nostra cucina. Abbiamo bisogno di bicchieri trasparenti di vetro o di cristallo. Se hanno una decorazione incisa, meglio ancora. Uno per uno, li mettiamo sotto una luce e osserviamo l'ombra che creano sul tavolo sottostante. Se guardiamo bene, osserviamo che ci sono zone più chiare nell'ombra. Queste sono zone dove le proprietà di rifrazione del bicchiere concentrano abbastanza luce da creare un punto luminoso. Se c'è una decorazione

incisa sul bicchiere, potremo riconoscere la decorazione nelle zone chiare dell'ombra. Queste zone chiare sono *riflessi proiettati causati dalla rifrazione*. Ci saranno anche zone illuminate del tavolo causate dalla luce riflessa dal vetro. Queste sono *riflessi proiettati* (caustics) *causati dalla riflessione*. Ora che sappiamo cosa stiamo cercando, troveremo questi riflessi in molte situazioni della vita di tutti i giorni : l'ombra proiettata da una lente di ingrandimento ne ha uno, la luce che filtra attraverso un acquario, la luce attraverso un pezzo di cellophane spiegazzato, eccetera. Li vediamo anche sul fondo di una piscina in una giornata luminosa. .Questi riflessi sono un effetto dell'illuminazione che può aumentare il realismo delle immagini renderizzate di questo genere di oggetti.
POV-Ray utilizza metodi che simulano solo i riflessi proiettati causati dalla rifrazione (quelli causati dalla riflessione non sono possibili da calcolare). Ci sono limitazioni nel processo di raytracing standard che lo rendono inadatto per certe applicazioni di simulazione dell'illuminazione, come prove ottiche e certe particolari applicazioni di progetti di illuminazione architettonica. I metodi che eseguono i calcoli, molto più complessi, necessari per avere una piena simulazione della luce, compresi i riflessi proiettati (come path-tracing, photon-tracing o raytracing bidirezionale) sono molto lenti e non pratici su piattaforme di media potenza. Questo significa che dobbiamo arrangiarci per ottenere i migliori risultati possibili, ma con un po'di esperimenti vedremo che è possibile simulare molto da vicino la realtà. Il metodo migliore, quando è possibile, è studiare un esempio dell'oggetto che stiamo pensando di renderizzare. Abbiamo bisogno di conoscere la forma dei riflessi che vogliamo ottenere e quindi modificare la nostra immagine fino a quando non siamo soddisfatti.

4.8.4.6.2 Applicare Riflessi ad una Scena.

I riflessi proiettati sono un nuovo tipo di finitura della superficie. Li applichiamo alle ombre di un oggetto trasparente e rifrangente aggiungendo la parola chiave `caustics` alla finitura. Proviamo, per cominciare, il seguente esempio (vedi il file **caustic1.pov**).

```
#include "colors.inc"
#include "textures.inc"

camera {
location <0, 15, -40>
look_at <-2, 0, 1>
angle 10
}

light_source { <10, 20, 10> color White }
// sotto mettiamo un piano uniforme per vedere le ombre
plane { y, 0
pigment { Grey }
}

// un oggetto per applicare i riflessi
sphere { <0, 3, 0>, 2
texture {
Glass3
finish { caustics .6 }
}
}
```

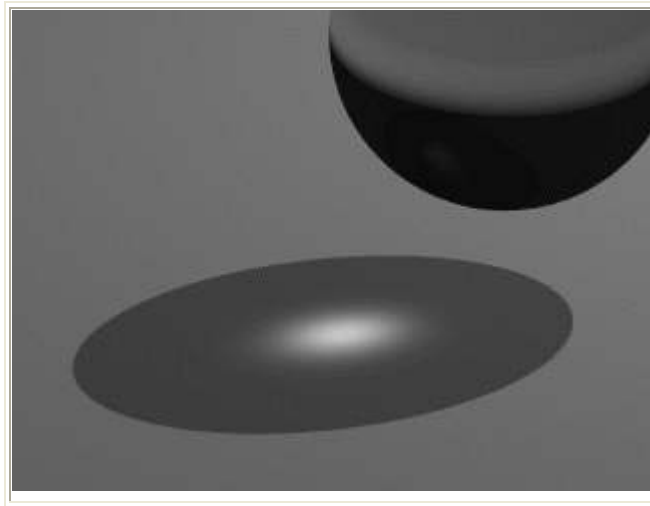



Fig. 136-Rilessi nell'ombra

Quando renderizziamo questo esempio, vediamo la nostra sfera nell'angolo superiore destro dell'immagine, un poco sopra il piano e vediamo l'ombra che proietta distribuita nella parte centrale dell'immagine. Nel centro, c'è un riflesso. L'area luminosa al centro dell'ombra rappresenta la luce che, normalmente, la rifrazione del vetro di cui la sfera è fatta concentrerebbe in quel punto. La sola domanda che rimane è : cos'è il valore decimale che segue la parola chiave `caustics` ?. Bene, è il punto in cui entra in scena il metodo di regolazione dei riflessi. Ricordi il bicchiere ? Se abbiamo un bicchiere con pareti sottili e una base di vetro spesso, vediamo il significato di questo parametro nell'ombra che il bicchiere proietta. In alto, essendo le pareti più sottili e quindi con minore rifrazione, i riflessi sono meno pronunciati e più diffusi nell'ombra, ma quando arriviamo alla parte di ombra generata dalla base, più spessa e più rifrangente, improvvisamente il riflesso diventa più pronunciato e più a fuoco vicino al centro.

Naturalmente, dato che questi sono riflessi simulati, non c'è corrispondenza tra quanto il riflesso è a fuoco e la forma, la dimensione e le caratteristiche di rifrangenza dell'oggetto. Ma possiamo controllarla manualmente con il valore decimale che segue la parola chiave `caustics`. Più questo valore si avvicina a zero, più sfumato e tenue diventa il riflesso. Più si avvicina ad 1 e più a fuoco e pronunciato diventa l'effetto. Ad un valore di 1, abbiamo il riflesso dato da un pezzo di cristallo al piombo, spesso ed altamente rifrangente, mentre a 0.1 sembra di più una sfera di vetro vuota all'interno. Proviamo queste differenza renderizzando la scena con valori che vanno da 0.1 a 1 e osserviamo le differenze.

Anche valori fuori da questa scala funzionano. Numeri maggiori di 1 danno riflessi sempre più a fuoco. Numeri minori di 1 sono piuttosto strani, ma interessanti. Essenzialmente, l'oggetto diventa illuminato in modi bizzarri e l'ombra diventa come un negativo fotografico di sé stessa. Qualcosa tipo pistola a raggi della fantascienza anni '50. Sembra strano, per nulla realistico, ma se vogliamo qualcosa di surreale vale la pena di provarlo e archiviare mentalmente il risultato, per eventuali usi futuri.

4.8.4.6.3 Riflessi e Normali

POV-Ray fa uso della perturbazione delle normali ad una superficie in un modo che è più singolare di quanto la gente normalmente non si accorga. Quando applichiamo una normale ad una texture, non modifichiamo in alcun modo la superficie, ma piuttosto, stiamo dicendo a POV-Ray di trattare la superficie come se fosse alterata, limitatamente allo scopo di calcolare l'illuminazione di ogni suo punto. In breve, è un trucco dato dalla luce e dall'ombra, che, supponendo che non stiamo osservando l'oggetto da un angolo troppo laterale, crea effettivamente l'illusione della distorsione sulla sua superficie. Anche i riflessi sono un trucco sintetico, come abbiamo visto prima e sono stati progettati per reagire a deformazioni della superficie come se queste deformazioni ci fossero

davvero. Ricordi l'esperimento del bicchiere di vetro ? Se abbiamo trovato un bicchiere con una decorazione incisa sulla sua superficie, probabilmente abbiamo notato che questa decorazione si vede anche nei riflessi proiettati dal vetro. Quando abbiamo una superficie trasparente con un blocco `normal{...}` applicato, i riflessi proiettati da questa superficie ne simulano le irregolarità, che si vedono poi nell'ombra che l'oggetto forma.
A seguire, un esempio di cosa intendiamo : semplicemente vogliamo rappresentare l'acqua in una piscina. Abbiamo ridotto questo esempio ad un piano sopra di noi, che rappresenta l'acqua, uno sotto che rappresenta il pavimento, una camera appena sotto il pelo dell'acqua, rivolta al pavimento ed una sorgente luminosa in alto (vedi il file **caustic2.pov**).

```
#include "colors.inc"
// La camera è sott'acqua, rivolta verso il fondo della piscina
// per vedere meglio i riflessi proiettati.

camera {
location <0, -5, 0>
look_at <0, -10, -5>
}

light_source { <0, 100, 49.5> color White }
// il fondo della piscina...
plane { y, -10
texture {
pigment { color rgb <0.6, 0.7, 0.7> }
finish { ambient 0.1 diffuse 0.7 }
scale 0.01
}
}

// e la superficie dell'acqua
plane { y, 0
texture {
pigment { rgbf <0.6, 0.67, 0.72, 0.9> }
normal {
bumps .6
scale <.75, .25, .25>
rotate <0, 45, 0>
}
finish { caustics .9 }
}
}
```

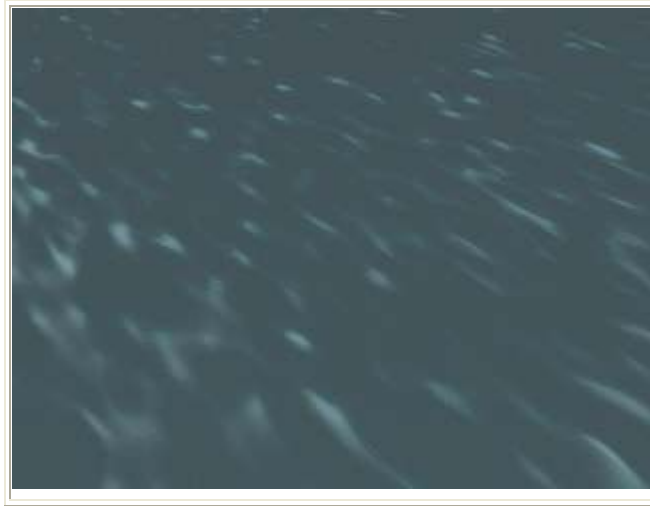



Fig. 137-Riflessi dell'acqua

Le irregolarità che abbiamo dato all'acqua con `bumps` rappresentano le ondulazioni piccole e casuali che si formano sull'acqua quando soffia una lieve brezza . Potremmo avere usato `ripples` o `waves`, come se qualcosa fosse caduto in acqua da poco, ma per un esempio, `bumps` va bene. Notiamo che il pavimento della piscina mostra decine di piccoli punti luminosi, che corrispondono approssimativamente alla superficie dell'acqua. Se vogliamo, possiamo mettere `ripples` o `waves` al posto di `bumps` e vedere come cambiano i riflessi sul fondo. Nonstante il fatto che un piano 'piatto' non proietti riflessi (possiamo provare eliminando il blocco `normal{...}`), il processo di generazione di falsi riflessi di POV-Ray 'sa' che se la superficie fosse realmente irregolare, come l'istruzione `normal{...}`indica, la luce rifratta dalla superficie sarebbe abbastanza per proiettare riflessi sul fondo della piscina. Vediamo che anche con una superficie curva, come la sfera dell'esempio precedente, la presenza di un blocco di istruzioni `normal{...}`causa il comparire di riflessi proiettati da un oggetto. Fatto abbastanza interessante, questo basterebbe per provare che i riflessi sono effettivamente falsi : la nostra acqua non ha nessuna proprietà di rifrazione specificata nella finitura, ma nonostante questo i riflessi sono esattamente gli stessi !

4.8.4.7 Iridescenza

L'iridescenza è ciò che vediamo su una bolla di sapone quando è colpita dalla luce del sole. L'effetto 'arcobaleno' è creato da un fenomeno detto 'interferenza da strato sottile' (vedi il paragrafo "Iridescenza", per maggiori dettagli). Per ora, proviamo ad usarla. L'iridescenza è specificata dalla parola chiave `irid` e da tre parametri : `amount`, `thickness` e `turbulence` (quantità, spessore e turbolenza). La quantità rappresenta il contributo dato dall'iridescenza al colore della superficie. Normalmente, bastano valori tra 0.1 e 0.5. Lo spessore influisce sulla complessità dell'effetto. I migliori risultati si ottengono tra 0.25 ed 1. La turbolenza è leggermente diversa da quella che abbiamo usato per i pigmenti e le normali. Non possiamo impostare `octaves`, `lambda`, o `omega` ma possiamo specificare un valore che influirà sulla complessità dell'effetto in un modo leggermente diverso dal valore dello spessore. Anche qui, i risultati migliori si ottengono con valori tra 0.25 ed 1. Infine, l'iridescenza sarà influenzata dalle irregolarità della superficie dato che dipende dall'angolo con cui i raggi di luce colpiscono la superficie. Sapendo tutto ciò, aggiungiamo un po' di iridescenza alla nostra sfera di vetro.

```
sphere { <0,0,0>, 1
pigment { White filter 1 }
finish {
ambient .1
diffuse .1
```

```
reflection .2
refraction 1
ior 1.5
specular 1
roughness .001
fade_distance 5
fade_power 1
caustics 1
irid {
0.35
thickness .5
turbulence .5
}
}
}
```

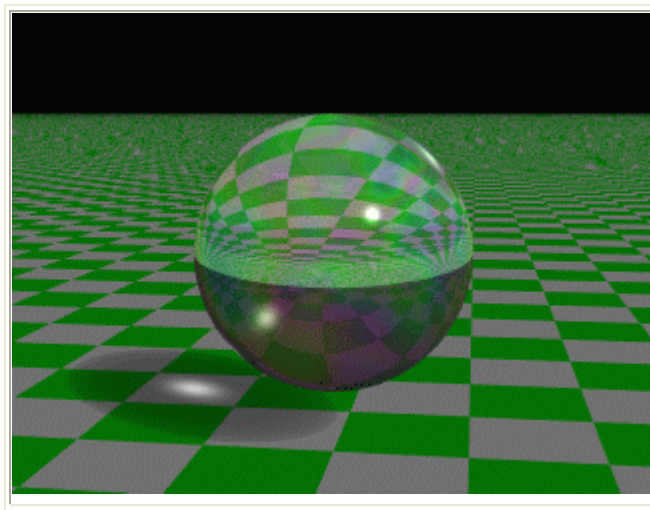



Fig. 138-Iridescenza

Proviamo a modificare i valori di `amount`, `thickness` e `turbulence` per vedere cosa cambia. Proviamo anche ad aggiungere un blocco `normal{...}` per vedere cosa succede.

4.8.5 Aloni

Nota Importante : gli aloni in POV-Ray 3.0 sono una funzione in qualche modo sperimentale. Ci sono forti probabilità che il design e l'implementazione di queste funzioni camberanno in versioni future del programma. Non possiamo garantire che le scene che utilizzano queste funzioni nella versione 3.0 potranno essere renderizzate nello stesso modo in versioni successive, o che la piena compatibilità del linguaggio sarà mantenuta.

Gli aloni (*halo*) sono una potente funzione che può essere utilizzata per creare molti effetti diversi come nuvole, fuoco, nebbia, fumo, laser, ecc. Il nome deriva dalla possibilità di renderizzare aloni simili a quelli che si vedono attorno al Sole o alla Luna.
A causa della complessità di questa funzione ed al grande numero di parametri disponibili, è molto difficile ottenere risultati soddisfacenti. I paragrafi successivi aiuteranno a creare un alone passo dopo passo, iniziando dagli elementi fondamentali e raffinando progressivamente il metodo.
E'anche utile leggere il paragrafo sugli aloni (§ 7.6.4) per comprendere meglio la funzione.
Dovrebbero anche, in particolare, essere letti i paragrafi "Oggetti Vuoti ed Oggetti Solidi" e "Mappatura degli Aloni", perché sono fondamentali per capire a fondo l'argomento.

4.8.5.1 Cosa Sono Gli Aloni ?

Gli aloni sono una funzione delle texture che ci permette di riempire l'interno di un oggetto con particelle. La distribuzione di queste particelle può essere modificata usando diverse funzioni di densità e di mappatura della densità. Le particelle possono emettere luce per dare effetti come fuoco o laser, o possono assorbire luce per creare nebbia o nuvole. Un alone è attaccato ad un oggetto, il cosiddetto contenitore, come se fosse una frase di specificazione di pigmento, normali, o finitura. L'oggetto contenitore è completamente riempito dall'alone, ma non possiamo vedere nulla se non ci assicuriamo che l'oggetto sia vuoto e che la superficie sia trasparente. Mostreremo nel prossimo paragrafo come ottenere ciò. Lavorando con gli aloni dobbiamo sempre ricordare che l'oggetto contenitore deve sempre essere vuoto e trasparente.

4.8.5.2 L'Alone Emittente

Iniziamo con uno dei tipi più semplici, l'alone emittente. Usa solo particelle che emettono luce. Non ci sono particelle che assorbono la luce emessa dalle altre particelle o da sorgenti luminose.

4.8.5.2.1 Iniziare con un Alone Semplice

Un approccio metodico nel creare un effetto di alone, è iniziare con un oggetto semplice, di dimensione unitaria, che si trova al centro del nostro sistema di riferimento. Nel primo esempio (file **halo01.pov**) cerchiamo di creare una potente esplosione, per cui l'oggetto migliore è la sfera. Iniziamo con una semplice scena costituita da una camera, una sorgente luminosa (non ci interessano le ombre, per cui aggiungiamo la parola chiave `shadowless`) un piano a scacchiera ed una sfera di raggio 1 contenente l'alone (*halo*).

```
camera {
location <0, 0, -2.5>
look_at <0, 0, 0>
}

light_source { <10, 10, -10> color rgb 1 shadowless }
plane { z, 2
pigment { checker color rgb 0, color rgb 1 }
finish { ambient 1 diffuse 0 }
scale 0.5
hollow
}

sphere { 0, 1
pigment { color rgbt <1, 1, 1, 1> }
halo {
emitting
spherical_mapping
linear
color_map {
[ 0 color rgbt <1, 0, 0, 1> ]
[ 1 color rgbt <1, 1, 0, 0> ]
}
samples 10
}
hollow
}
```

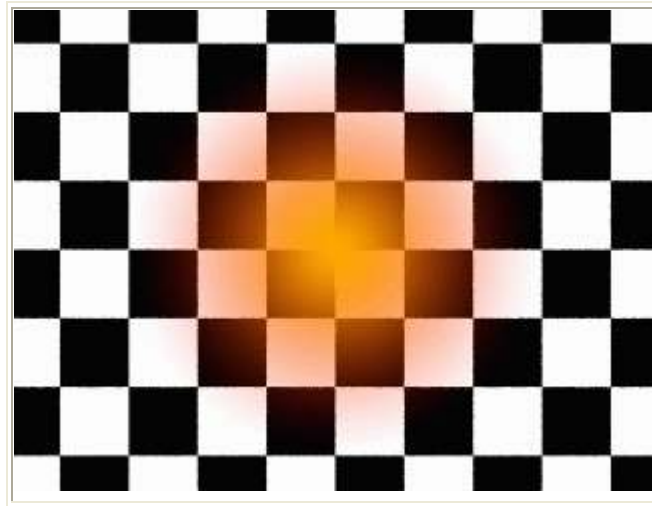



Fig. 139-Semplice alone

Notiamo che la sfera è impostata per essere vuota ed ha la superficie completamente trasparente (il canale della trasmittanza nel colore è impostato ad 1), proprio come è richiesto per le halo. Notiamo anche che il piano ha la parola chiave `hollow` nonostante che non abbia associato nessun alone. Perché è necessario ?
Il motivo è molto semplice. Come si spiega nel paragrafo "Oggetti Vuoti e Oggetti Solidi", non ci può essere nessuna halo all'interno di un oggetto non vuoto. Dato che la camera si trova all'interno del piano, dalla parte del piano, cioè che è considerata essere 'dentro', l'alone non sarebbe visibile a meno che il piano non sia reso vuoto specificando l'attributo `hollow` (cavo) (oppure, si aggiunga la parola chiave `negative` per portare la camera dal lato positivo del piano).
Cosa significano tutte queste parole chiave e valori ? All'inizio della halo, la parola chiave `emitting` è usata per specificare che tipo di halo si vuole definire. La halo emittente (emitting) emette luce. Questa è la soluzione migliore per la nostra esplosione. Le parole chiave `spherical_mapping` e `linear` hanno bisogno di una spiegazione più dettagliata sul funzionamento delle halo (questa si trova, in maggiore dettaglio, nel paragrafo "Halo").
Come abbiamo notato sopra, un alone è formato da una grande quantità di piccole particelle. La distribuzione nello spazio di queste particelle è determinata da una funzione di densità. In generale, una funzione di densità ci dice quante particelle possiamo trovare in una data posizione nello spazio. Invece di utilizzare una funzione di densità esplicita, definita matematicamente, le halo si basano su di un sistema di mappature di densità e di funzioni di densità per modellare una varietà di distribuzioni di particelle. Il primo passo in questo modello è la funzione di mappatura della densità che viene utilizzata per mappare punti a tre dimensioni (di uno spazio tridimensionale) in un set di valori monodimensionali. Nel nostro esempio, utilizziamo una mappatura sferica, cioè consideriamo la distanza di un punto dal centro del sistema di coordinate. Questo è il motivo per cui è meglio iniziare con un oggetto contenitore che si trova al centro del sistema di coordinate. Dato che tutte le mappature di densità sono calcolate rispetto al centro del sistema di assi, non vedremmo nulla, se iniziassimo con un oggetto contenitore che si trovasse da qualunque altra parte. Il metodo corretto di posizionare un oggetto contenitore, è spostare tutto l'oggetto dopo averlo definito (comprese texture e halo).
Ora abbiamo per la distanza dal centro, un valore compreso tra 0 ed 1. Questo valore verrà trasformato usando una funzione densità per ottenere valori di densità invece che valori di distanza. Usare questo singolo valore non funziona perché vogliamo avere una distribuzione della densità delle particelle che diminuisca man mano che ci allontaniamo dal centro dell'oggetto contenitore verso l'esterno. Ciò avviene grazie alla funzione densità. Ci sono diverse alternative disponibili nel paragrafo di riferimento sulle halo (vedi § 7.6.4 e "Funzione Densità"). Usiamo la più semplice funzione lineare che mappa i valori tra 0 ed 1 in un raggio di valori tra 1 e 0. Quindi, otteniamo una

densità di 1 al centro della sfera e di 0 ad una distanza dal centro uguale al raggio.
Ora che abbiamo una funzione densità, cosa dobbiamo fare per vedere qualcosa ? Qui entra in gioco la parola chiave `colour_map`. Viene usata per descrivere una mappatura di colore che in effetti dice al programma quali colori vengono usati in corrispondenza di una certa densità. La relazione è semplice : i colori all'inizio della mappa di colore (con valori bassi) saranno usati nelle zone con bassi valori di densità e i colori alla fine della mappa (con alti valori) saranno usati per le zone di alta densità. Nel nostro esempio, la halo sarà gialla al centro, dove la densità è maggiore e sfumerà al rosso verso la superficie della sfera, dove la densità si avvicina allo 0.
Il canale della trasmittanza dei colori nella mappa è usato per modellare la trasparenza del campo di densità. Un valore di 0 rappresenta la completa opacità, cioè aree con la densità corrispondente saranno (quasi) opache, mentre il valore di 1 significa (quasi) totale trasparenza. Nel nostro esempio, usiamo

```
color_map {
[ 0 color rgbt <1, 0, 0, 1> ]
[ 1 color rgbt <1, 1, 0, 0> ]
}
```

Che risulta in una halo con una zona esterna molto trasparente, rossastra ed una zona interna quasi opaca e gialla.
C'è un altro parametro che deve ancora essere spiegato : la parola chiave `samples`. Questa parola chiave dice a POV-Ray quanti campioni devono essere presi lungo ogni raggio che attraversa la halo per calcolare il suo effetto. Usare un basso valore porterà ad un'alta velocità del rendering, mentre un alto valore darà tempi di rendering considerevolmente più lunghi. Il numero di campioni deve essere aumentato quando l'alone ha un aspetto 'confuso', cioè quando risultano evidenti alcune irregolarità dovute al basso tasso di campionamento. Per maggiori dettagli, vedi il paragrafo "Campionamento per gli Aloni".
Un buon valore iniziale per il numero di campioni è 10.

4.8.5.2.2 Aumentare la Luminosità

I colori dell'alone nell'immagine che abbiamo appena visto sono piuttosto tenui. Si vede troppo sfondo dietro l'alone. Non assomiglia molto a del fuoco, vero ? Un modo semplice per rimediare a questo inconveniente, è diminuire la trasparenza delle particelle nelle zone ad alta densità. Otteniamo questo risultato usando la seguente mappa di colori al posto della vecchia (il valore di trasmittanza negativo è corretto).

```
color_map {
[ 0 color rgbt <1, 0, 0, 1> ]
[ 1 color rgbt <1, 1, 0, -1> ]
}
```

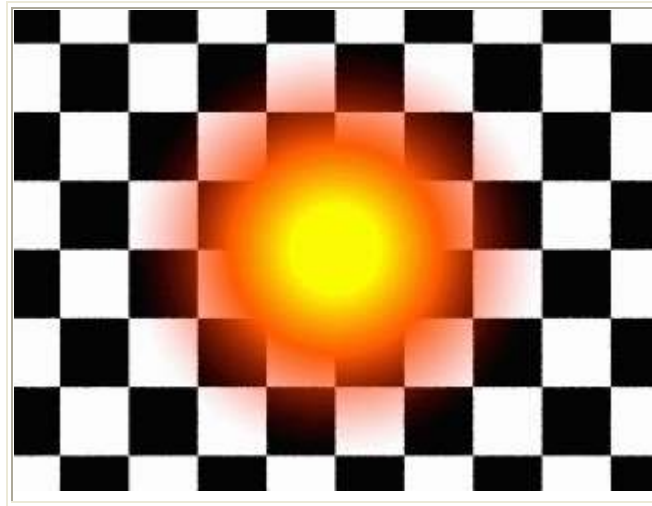



Fig. 141-Aumentare la luminosità

Osservando il risultato (file **halo02.pov**) vediamo che l'alone è senza dubbio molto più luminoso.

4.8.5.2.3 Aggiungere Turbolenza

Ciò che abbiamo ottenuto non assomiglia ancora ad un'esplosione. E' più una palla brillante. Vogliamo renderla più caotica, in qualche modo, aggiungerle un po' di turbolenza. E' possibile usando la parola chiave `turbulence` insieme alla quantità di turbolenza che vogliamo aggiungere. Esattamente come nell'esempio seguente :

```
sphere { 0, 1
pigment { color rgbt <1, 1, 1, 1> }
halo {
emitting
spherical_mapping
linear
turbulence 1.5
color_map {
[ 0 color rgbt <1, 0, 0, 1> ]
[ 1 color rgbt <1, 1, 0, -1> ]
}
samples 10
}
hollow
}
```

Fig. 142-Gli aloni e la turbolenza

Aggiungere turbolenza alla halo sposta tutti i punti all'interno del contenitore in maniera pseudo-casuale. Ciò risulta in una distribuzione delle particelle che fa sembrare che ci sia una specie di flusso nella halo (ed in dipendenza di quanto alziamo il valore di `turbulence` otteniamo un flusso laminare o turbolento). Usiamo un valore alto perché un'esplosione è molto turbolenta. Guardando l'immagine di esempio (vedi **halo03.pov**) vedremo che ora la halo assomiglia molto di più ad un'esplosione della palla che avevamo finora. Notiamo che il tempo di rendering è aumentato dopo che abbiamo aggiunto la turbolenza. Questo è dovuto al fatto che per ogni campione preso dalla halo deve essere calcolata la funzione turbolenza.

4.8.5.2.4 Ridimensionare l'Alone

Notiamo un fatto strano, rispetto alla nostra esplosione. Assomiglia ancora ad una sfera. Perché succede questo e come possiamo evitarlo ?
Come abbiamo notato prima, aggiungere la turbolenza sposta le particelle all'interno del contenitore. Il problema è che alcune delle particelle si sono spostate all'esterno del contenitore e questo porta a zone di alta densità sulla superficie dell'oggetto contenitore, rivelandone la forma (le particelle fuori dal contenitore sono perdute, diventano invisibili e quindi vediamo un brusco cambio della densità in coincidenza della superficie del contenitore).
Un semplice metodo di evitare questo inconveniente è assicurarsi che le particelle restino all'interno del contenitore anche se aggiungiamo un po' di `turbulence`. Otteniamo questo risultato diminuendo le dimensioni della halo (non il contenitore, solo la halo). Aggiungiamo quindi un comando `scale` all'interno della frase `halo{...}`.

```
sphere { 0, 1
pigment { color rgbt <1, 1, 1, 1> }
halo {
emitting
spherical_mapping
linear
turbulence 1.5
color_map {
[ 0 color rgbt <1, 0, 0, 1> ]
[ 1 color rgbt <1, 1, 0, -1> ]
}
samples 10
scale 0.5
```

```
}
hollow
scale 1.5
}
```

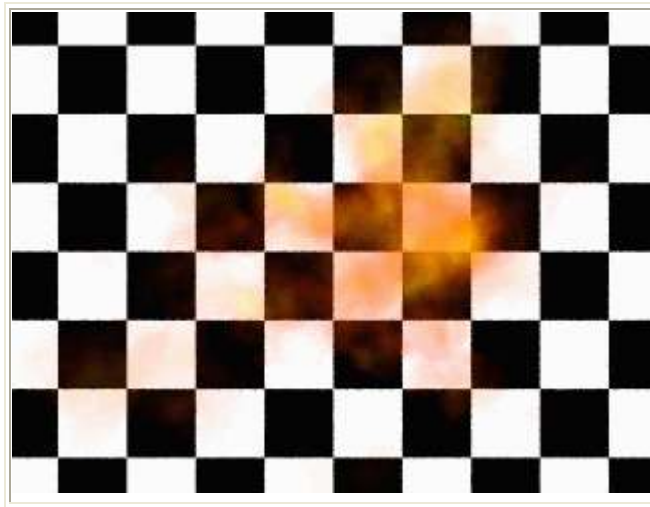



Fig.143-Ridimensionare l'alone

Il comando `scale 0.5` dice a POV-Ray di ridimensionare tutti i punti all'interno del contenitore di questa quantità. In effetti, questo ridimensiona il raggio che otteniamo dopo che il programma ha eseguito la mappatura di densità da 0 a 0.5 invece che da 0 ad 1 (senza considerare l'effetto dato dalla turbolenza).
Se ora aggiungiamo la turbolenza, i punti della halo possono muoversi di mezza unità in ogni direzione, senza abbandonare il contenitore. E' esattamente quello che vogliamo. Per compensare la halo più piccola, aumentiamo le dimensioni della sfera (e della halo in essa contenuta) di un fattore 1.5.

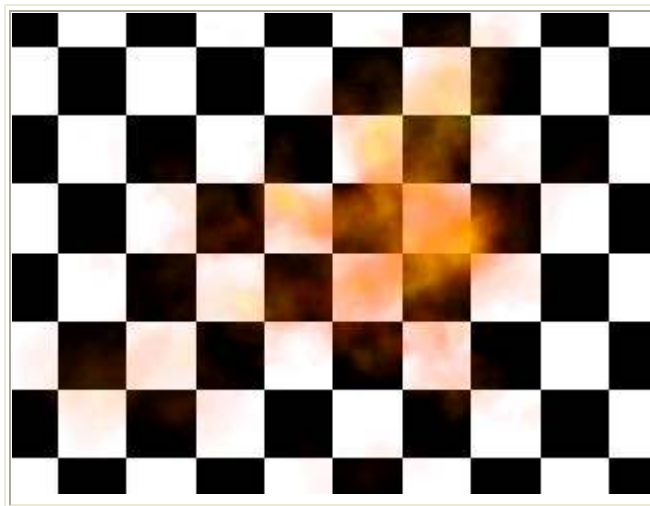



Fig. 144-Esplosione

Guardando la nuova immagine (**halo04.pov**) non vediamo più traccia della sfera che contiene la halo. Finalmente abbiamo ottenuto una bella esplosione.
La quantità di cui ridimensioniamo la halo dipende da quanta turbolenza usiamo. Maggiore è la turbolenza, più la halo deve essere rimpicciolita. Questo è un argomento sul quale sperimentare. Un altro modo per evitare che i punti della halo escano dalla sfera, è di usare una sfera più grande, cioè una sfera con un raggio maggiore di uno. E' importante ridimensionare la sfera prima di aggiungere la halo perché altrimenti anche la halo verrà ingrandita.

Notiamo che questo funziona solo per mapping sferico e cubico (e per funzioni densità non costanti). Tutte le altre funzioni di mappatura sono infinite, cioè la distribuzione delle particelle copre uno spazio infinito. (vedi anche "Mappatura della Densità").

4.8.5.2.5 Usare la Frequenza per Migliorare il Realismo

Un altro metodo molto buono per migliorare il realismo della nostra esplosione, è usare un valore per la frequenza (parola chiave `frequency`) diverso da 1. Il modo in cui il parametro `frequency` funziona è spiegato nel paragrafo "Il Modificatore Frequency (Frequenza)".
La spiegazione, piuttosto matematica che viene data in quel paragrafo, non ci aiuta però a capire come usare la frequenza. Nonostante tutto, è molto semplice. Il valore assegnato alla frequenza dice al programma quante volte la mappatura del colore sarà ripetuta nel raggio da 0 a 1. Se viene specificata una frequenza di 1 (il valore predefinito), la mappatura del colore specificata sarà ripetuta una volta nel campo delle densità, cioè, il colore al valore 0 sarà usato per la densità 0, il colore al valore 0.5 sarà usato alla densità 0.5 e così via. Semplice, no ?
Se scegliamo invece una frequenza 2, il colore a 0 sarà usato per la densità 0, il colore a 0.5 sarà usato per la densità 0.25 ed il colore a 1 sarà usato per la densità 0.5. Cosa succede alle densità maggiori di 0.5 ? Dato che non ci sono colori per valori superiori ad 1, ricominciamo dallo 0. Quindi, il colore a 0.1 sarà riutilizzato per la densità 0.55, ((2*0.55)mod 1=1.1 mod 1 = 0.1) il colore a 0.5 sarà riutilizzato per la densità 0.75 ed il colore a 1 nella nostra mappatura, sarà usato per la densità 1.
Se siamo dei bravi matematici, noteremo che l'esempio sopra è errato, perché (1*2)mod 1 = 0 e non 1. Pensiamo di usare un valore appena più piccolo di 1 e tutto funzionerà.
Possiamo avere notato che, per evitare bruschi cambiamenti di colore nelle halo definite con una frequenza maggiore di 1, dovremo usare delle mappature per il colore periodiche, cioè, che abbiano lo stesso colore a 0 ed 1.
Cambiamo il nostro esempio usando una mappatura periodica del colore e cambiando il valore della frequenza a 2.

```
sphere { 0, 1
pigment { color rgbt <1, 1, 1, 1> }
halo {
emitting
spherical_mapping
linear
turbulence 1.5
color_map {
[ 0.0 color rgbt <1, 0, 0, 1> ]
[ 0.5 color rgbt <1, 1, 0, -1> ]
[ 1.0 color rgbt <1, 0, 0, 1> ]
}
frequency 2
samples 20
scale 0.5
}
hollow
scale 1.5
}
```

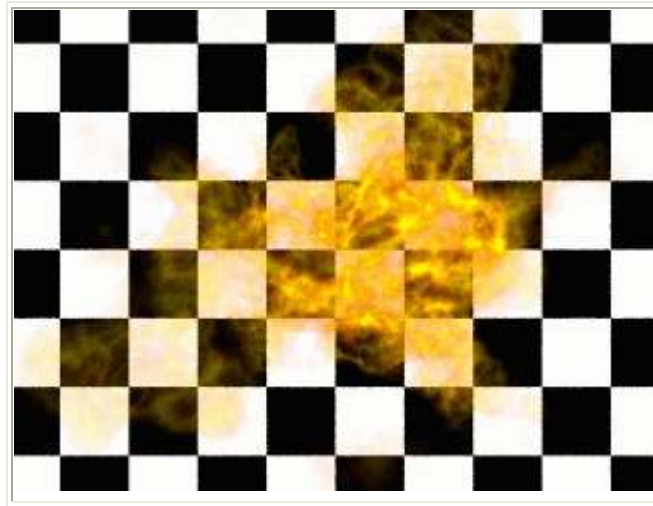



Fig. 145-Esplosione

Osservando il risultato (vedi **halo05.pov**) possiamo dirci soddisfatti dell'esplosione che abbiamo fatto, no ? C'è un'altra cosa alla quale dovremmo fare attenzione quando aumentiamo il valore della frequenza. Spesso è necessario aumentare il numero di campioni (`samples`) quasi allo stesso modo in cui cambiamo la frequenza. Se non lo facciamo, probabilmente otterremo delle gravi artificiosità (come 'salti' di colore, o strane strisce colorate). Se ciò avviene, cambia il numero di campioni in accordo al valore della frequenza (il doppio dei campioni per un valore di frequenza raddoppiato, ad esempio).

4.8.5.2.6 Cambiare il Colore dell'Alone

Abbiamo una bella esplosione, ma vogliamo aggiungerle un tocco di fantascienza usando colori diversi. Che ne diresti di un'esplosione rossa, meno turbolenta, che diventa verde sui bordi ? Niente di più facile !

```
sphere { 0, 1.5
pigment { color rgbt <1, 1, 1, 1> }
halo {
emitting
spherical_mapping
linear
turbulence 0.5
color_map {
[ 0 color rgbt <0, 1, 0, 1> ]
[ 1 color rgbt <1, 0, 0, -1> ]
}
samples 10
scale 0.75
}
hollow
scale 1.5
}
```

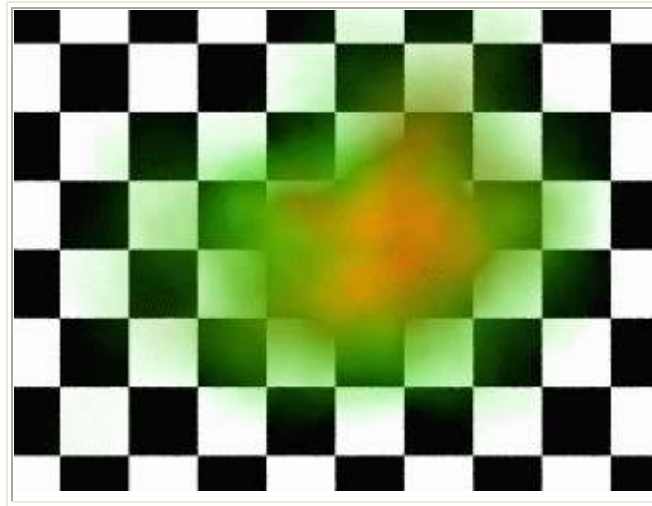



Fig. 146-Cambiare colore

Così dovrebbe funzionare, ma guardando il risultato di **halo06.pov** potremmo esserne delusi. Dov'è il centro dell'esplosione ? I bordi sono verdi, come ci aspettavamo, ma c'è molto giallo nel centro e solo un po' di rosso. Cosa succede ?
Stiamo usando un alone emittente. Secondo il corrispondente paragrafo ("Alone Emittente"), questo tipo di alone utilizza particelle molto piccole che non attenuano la luce che passa attraverso la halo. Soprattutto, le particelle vicine al punto di osservazione non attenuano la luce che proviene dalle particelle lontane dall'osservatore. Durante il calcolo del colore della halo vicino al centro della sfera (contenitore), i raggi attraversano quasi tutta la gamma delle densità della distribuzione di particelle. Quindi, otteniamo un colore sia rosso che verde, man mano che il raggio procede, a seconda della posizione nella halo. La somma di questi due colori è il giallo e per questo vediamo giallo il centro della halo.
Come possiamo ottenere quello che vogliamo ? La risposta è : usando una halo tralucente anziché una emittente. La halo tralucente è molto simile a quella emittente, con la differenza che attenua la luce che la attraversa. In questo modo, la luce proveniente dalle particelle che si trovano dietro rispetto all'osservatore, sarà attenuata da quella delle particelle che stanno davanti.
Per una spiegazione sull'alone tralucente, vedi il paragrafo "Alone Tralucente" .

4.8.5.3 Alone Tralucente

Abbiamo nominato l'alone tralucente nel paragrafo sull'alone emittente, come un modo per evitare il mescolamento dei colori. L'alone tralucente è molto simile all'emittente, con la differenza che assorbe anche la luce. Possiamo vederlo come una combinazione dell'alone emittente e di quello attenuante (descritto nel paragrafo "Alone Attenuante").
Sostituendo la parola chiave `emitting` dell'esempio nel paragrafo "Cambiare il Colore dell'Alone", con la parola chiave `glowing`, otteniamo l'effetto desiderato (vedi **halo11.pov**).

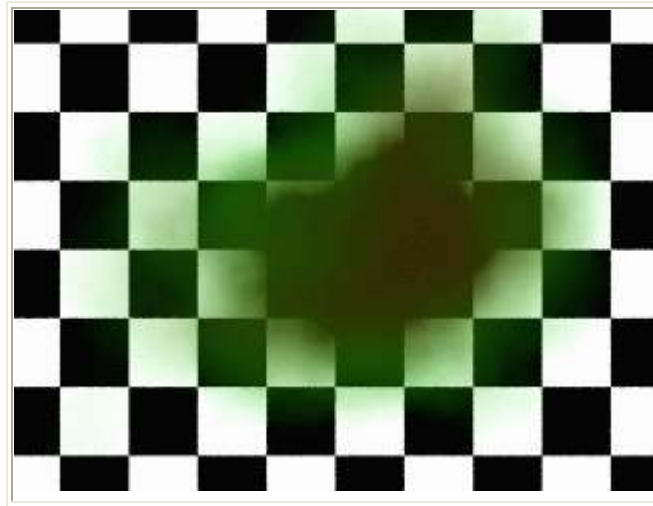



Fig. 147-Alone tralucente

Anche se il rosso delle zone ad alta densità non è molto visibile, perché le zone verdi, a minore densità, che si trovano davanti assorbono la maggior parte della luce rossa, non abbiamo più il centro della halo giallo.
A causa della sua somiglianza con la halo emittente, ci conviene fare qualche esperimento con questo tipo di halo. Dobbiamo solo ricordare tutto ciò che abbiamo imparato nelle precedenti sezioni per ottenere qualche risultato soddisfacente.

4.8.5.4 Alone Attenuante

Un altro semplice tipo di alone è l'alone attenuante, che assorbe luce. Una profonda differenza tra l'alone attenuante e gli altri tipi, è che il colore dell'alone viene calcolato a partire dalla mappatura di colore usando la densità *totale* delle particelle lungo un dato raggio. Gli altri tipi di alone calcolano una media pesata dei colori ricavati dalla densità per ogni campione.

4.8.5.4.1 Fare una Nuvola

Le halo attenuanti sono perfette per creare nuvole e fumo. Negli esempi seguenti cercheremo di fare una nuvoletta. Ripartiamo da una sfera di dimensione unitaria, riempita con una semplice halo attenuante (vedi **halo21.pov**).

```
camera {
location <0, 0, -2.5>
look_at <0, 0, 0>
}

light_source { <10, 10, -10> color rgb 1 shadowless }
plane { z, 2
pigment { checker color rgb 0, color rgb 1 }
finish { ambient 1 diffuse 0 }
scale 0.5
hollow
}

sphere { 0, 1
pigment { color rgbt <1, 1, 1, 1> }
halo {
attenuating
```

```
spherical_mapping
linear
color_map {
[ 0 color rgbt <1, 0, 0, 1> ]
[ 1 color rgbt <1, 0, 0, 0> ]
}
samples 10
}
hollow
}
```

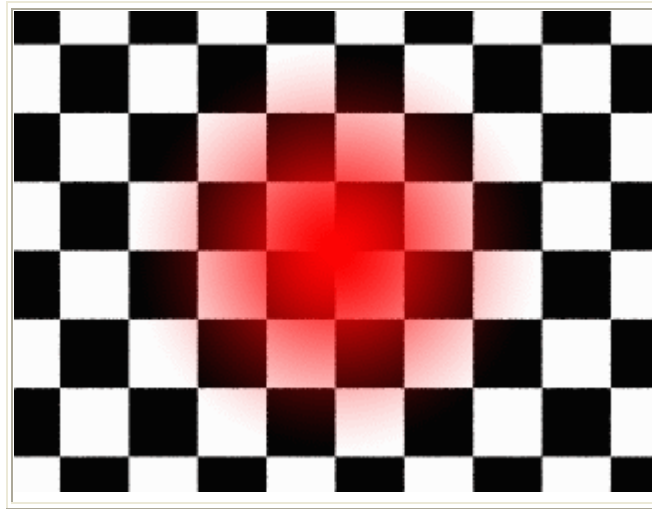



Fig. 148-Primo elemento: alone semplice

Sebbene le nuvole siano normalmente bianche, o grigie, ma non rosse, faremo questa nuvola rossa per renderla più visibile contro lo sfondo a scacchi bianchi e neri.
Il colore della halo attenuante, è calcolato dalla densità totale accumulata dopo che un raggio ha 'viaggiato' attraverso tutto il campo di particelle. Dobbiamo ricordarci di questo quando creiamo la mappa dei colori. Vogliamo che le zone a bassa densità della nuvola siano altamente trasparenti, per cui usiamo un colore rgbt <1,0,0,1> e vogliamo altresì che le zone ad alta densità siano opache e quindi scegliamo rgbt <1,0,0,0>.

4.8.5.4.2 Ridimensionare il Contenitore dell'Alone

La nuvola che abbiamo fatto prima non sembra molto realistica. E' solo una palla rossa, parzialmente trasparente. Per ottenere un risultato migliore, dobbiamo usare qualcuno dei metodi che abbiamo imparato nei paragrafi precedenti sulle halo emittenti. Aggiungiamo un po' di turbolenza per ottenere una sagoma più realistica, ingrandiamo il contenitore dell'alone per evitare che la sua superficie diventi visibile e diminuiamo la trasparenza delle aree con alta densità.
Un'altra idea è dimensionare il contenitore in modo da ottenere un'ellissoide che può essere usato per dare forma ad una nuvola. Possiamo farlo col comando `scale <1.5,0.75,1>` in fondo alla frase `sphere{...}`. In questo modo, vengono ridimensionati sia la sfera che il contenitore.

```
sphere { 0, 1
pigment { color rgbt <1, 1, 1, 1> }
halo {
attenuating
spherical_mapping
linear
turbulence 1
```

```
color_map {
[ 0 color rgbt <1, 0, 0, 1> ]
[ 1 color rgbt <1, 0, 0, -1> ]
}
samples 10
scale 0.75
}
hollow
scale <1.5, 0.75, 1>
}
```

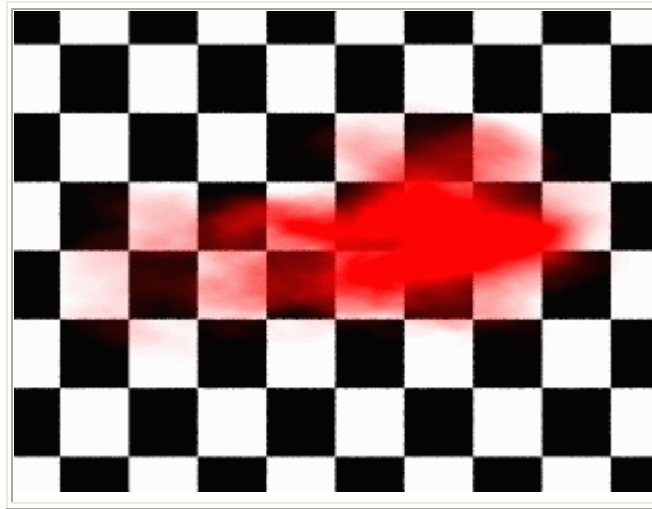



Fig. 149-Comincia ad assomigliare ad una nuvola

Osservando il risultato di **halo22.pov** vediamo che questa sembra più una vera nuvola (a parte il colore).

4.8.5.4.3 Aggiungere Aloni Supplementari

Un altro sistema per ottenere maggiore realismo è utilizzare aloni multipli. Se guardiamo le nubi a cumulo, per esempio, notiamo che si estendono spesso verso l'alto, mentre sono quasi piatte in basso. Vogliamo modellare questo aspetto aggiungendo due halo supplementari al nostro oggetto contenitore (vedi il paragrafo "Aloni Multipli", per maggiori dettagli) nel modo seguente :

```
sphere { 0, 1.5
pigment { color rgbt <1, 1, 1, 1> }
halo {
attenuating
spherical_mapping
linear
turbulence 1
color_map {
[ 0 color rgbt <1, 0, 0, 1> ]
[ 1 color rgbt <1, 0, 0, -1> ]
}
samples 10
scale <0.75, 0.5, 1>
translate <-0.4, 0, 0>
}
halo {
```

```
attenuating
spherical_mapping
linear
turbulence 1
color_map {
[ 0 color rgbt <1, 0, 0, 1> ]
[ 1 color rgbt <1, 0, 0, -1> ]
}
samples 10
scale <0.75, 0.5, 1>
translate <0.4, 0, 0>
}
halo {
attenuating
spherical_mapping
linear
turbulence 1
color_map {
[ 0 color rgbt <1, 0, 0, 1> ]
[ 1 color rgbt <1, 0, 0, -1> ]
}
samples 10
scale 0.5
translate <0, 0.2, 0>
}
hollow
}
```

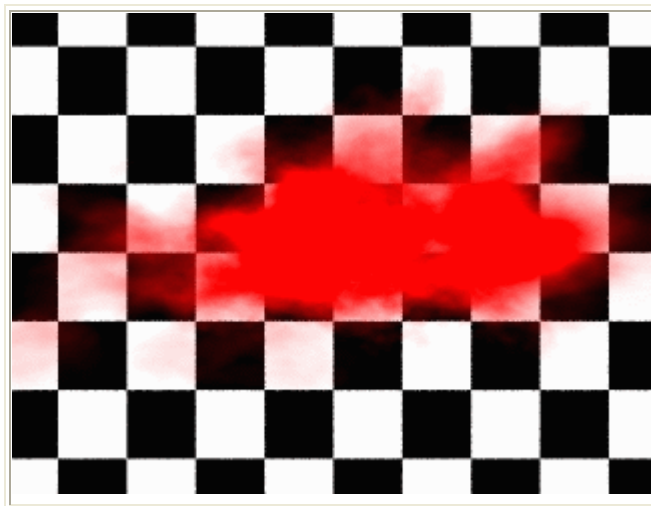



Fig. 150-Tre halo

Le tre halo utilizzate differiscono solo nella posizione, cioé nel vettore di traslazione che abbiamo usato. Le prime due halo servono a formare la base della nuvola, mentre la terza si trova sopra le altre. La sfera ha un raggio diverso da quella che abbiamo usato prima perché c'è bisogno di più spazio per tutte e tre le halo.
Il risultato di **halo23.pov** assomiglia in qualche modo ad una nuvola, per quanto possa avere bisogno di qualche rifinitura.

4.8.5.5 L'Alone Polvere

L'alone polvere è un tipo di alone molto complesso. Ci permette di vedere l'interazione della luce che proviene da una sorgente luminosa con le particelle dell'alone. Queste particelle assorbono luce nello stesso modo dell'alone attenuante, in più disperdono la luce. Questo rende visibili i raggi di luce e le ombre proiettate dagli oggetti

4.8.5.5.1 Iniziare con un Oggetto Illuminato da uno Spot

Iniziamo da un oggetto di forma rettangolare illuminato da una luce spot. Non usiamo nessuna halo al momento perché vogliamo vedere se l'oggetto è completamente illuminato dalla luce (file **halo31.pov**).

```
camera {
location <0, 0, -2.5>
look_at <0, 0, 0>
}

background { color rgb <0.2, 0.4, 0.8> }
light_source {
<2.5, 2.5, -2.5>
colour rgb <1, 1, 1>
spotlight
point_at <0, 0, 0>
radius 12
falloff 15
tightness 1
}

difference {
box { -1, 1 }
box { <-1.1, -0.8, -0.8>, <1.1, 0.8, 0.8> }
box { <-0.8, -1.1, -0.8>, <0.8, 1.1, 0.8> }
box { <-0.8, -0.8, -1.1>, <0.8, 0.8, 1.1> }
pigment { color rgb <1, 0.2, 0.2> }
scale 0.5
rotate 45*y
rotate 45*x
}
```

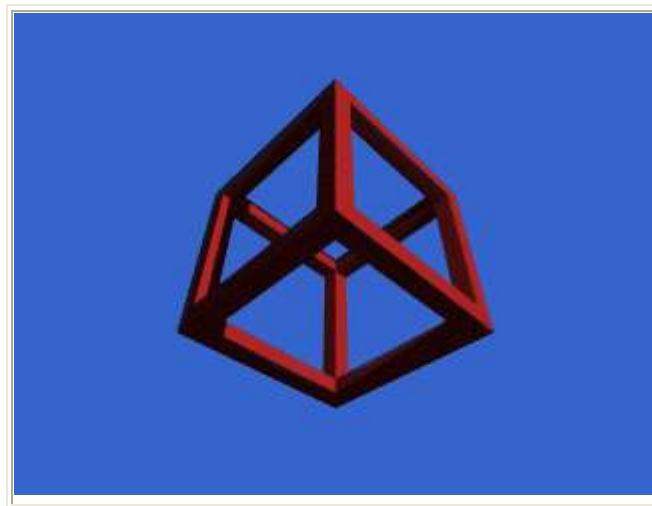



Fig. 151-Oggetto

Come possiamo vedere, l'oggetto è illuminato dalla sorgente luminosa. Ora possiamo iniziare ad aggiungere un po' di polvere.

4.8.5.5.2 Aggiungere Polvere

Useremo un parallelepipedo come contenitore per la nostra halo polvere. Dato che usiamo una funzione di densità costante, non importa il *tipo* di densità che useremo. La densità ha il valore specificato dalla parola chiave `max_value` in ogni punto della halo (il valore predefinito è 1). La dispersione isotropica è selezionata con la parola chiave `dust_type`.

```
box { -1, 1
pigment { colour rgbt <1, 1, 1, 1> }
halo {
dust
dust_type 1
box_mapping
constant
colour_map {
[ 0 color rgbt <1, 1, 1, 1> ]
[ 1 color rgbt <1, 1, 1, 0> ]
}
samples 10
}
hollow
scale 5
}
```

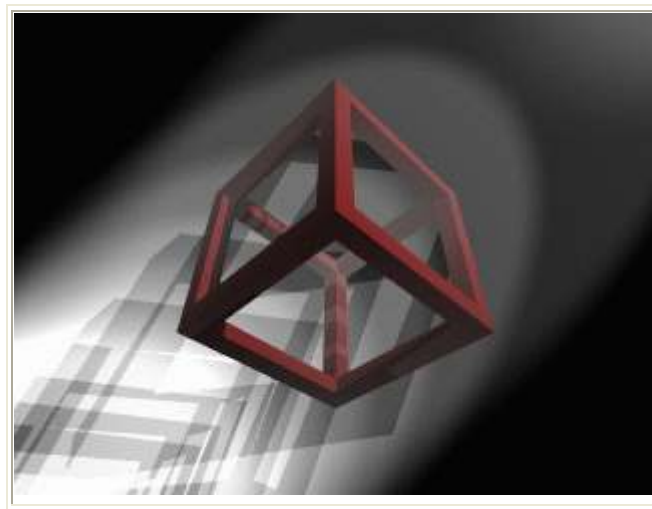



Fig. 152-Aggiungere un pò di polvere

Il risultato (vedi **halo32.pov**) è troppo luminoso. La polvere è troppo fitta e possiamo vedere solo parti dell'oggetto e nulla dello sfondo.

4.8.5.5.3 Diminuire la Densità della Polvere

La densità all'interno della halo ha il valore costante 1. Questo significa che solo il colore corrispondente al valore 1 della mappatura del colore viene usato per determinare densità e colore della polvere.
Utilizziamo un valore di trasmittanza di 0.7 in modo da ottenere una polvere più rarefatta.

```
box { -1, 1
```

```
pigment { colour rgbt <1, 1, 1, 1> }
halo {
dust
dust_type 1
box_mapping
constant
colour_map {
[ 0 color rgbt <1, 1, 1, 1.0> ]
[ 1 color rgbt <1, 1, 1, 0.7> ]
}
samples 10
}
hollow
scale 5
}
```

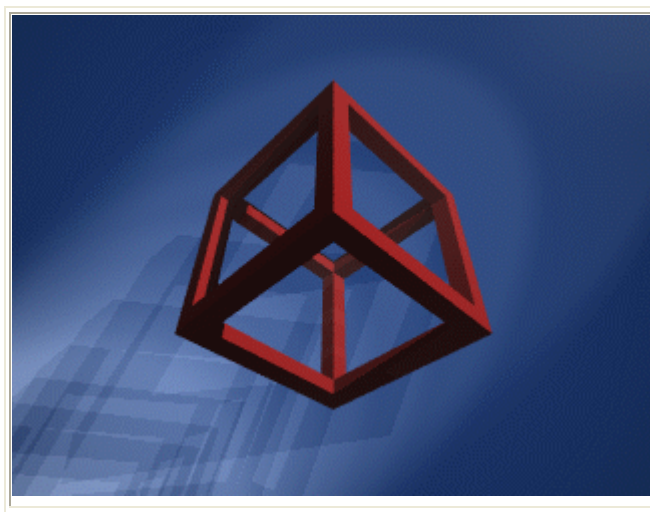



Fig. 153-Maggiore trasparenza alla polvere

A parte le brutte 'scalettature', l'immagine è molto migliorata. Possiamo vedere tutto l'oggetto e si intravede anche lo sfondo. (vedi **halo33.pov**)

4.8.5.5.4 Migliorare le Ombre

Per ridurre le artificiosità dovute all'aliasing utilizzeremo tre tecniche diverse : dispersione (jittering), sovracampionamento (super-sampling) ed un tasso di campionamento più alto in generale. La dispersione viene utilizzata per aggiungere casualità ai punti di campionamento, rendendo l'immagine più sfumata. Questo aiuta, dato che le scalettature regolari dovute all'aliasing si notano di più di un po'di rumore casuale. Un basso valore per `jitter` è la scelta migliore.
Il sovracampionamento è utilizzato per rilevare i particolari più fini lanciando raggi supplementari nelle zone dove ci sono molti cambiamenti in poco spazio (ad esempio, una brusca variazione di colore come può essere il bordo di un oggetto chiaro contro uno sfondo scuro). La soglia alla quale interviene il sovracampionamento e il massimo livello di ricorsività possono essere specificati usando le parole chiave `aa_threshold` e `aa_level`, rispettivamente.
L'approccio che funziona sempre è quello di aumentare il tasso di campionamento su tutta la scena. Dato che questo è anche il metodo più lento, dovremmo provare prima gli altri due e se non bastano, ricorrere a quest'ultimo.
Usiamo la halo seguente per ridurre gli artefatti di aliasing (**halo34.pov**).

```
box { -1, 1
```

```
pigment { colour rgbt <1, 1, 1, 1> }
halo {
dust
dust_type 1
box_mapping
constant
colour_map {
[ 0 color rgbt <1, 1, 1, 1.0> ]
[ 1 color rgbt <1, 1, 1, 0.7> ]
}
samples 50
aa_level 3
aa_threshold 0.2
jitter 0.1
}
hollow
scale 5
}
```

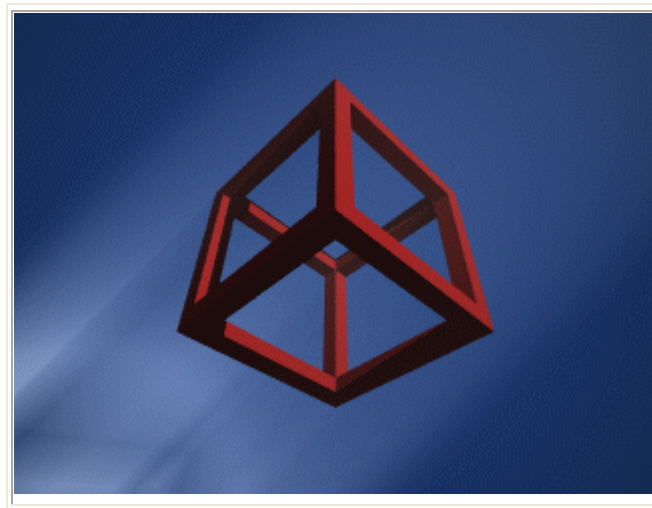



Fig. 154-Aggiungere un pò di dispersione de colore

L'immagine è molto migliore, ora. Artificiosità dovute all'aliasing si vedono appena.
I parametri che abbiamo usato ora sono discussi nel paragrafo sull'atmosfera (vedi "L'Atmosfera") per maggiori dettagli.

4.8.5.5.5 Aggiungere Turbolenza

La principale differenza tra la polvere che abbiamo appena fatto e l'atmosfera descritta nel paragrafo "L'Atmosfera" è la possibilità di scegliere una distribuzione di particelle non uniforme per la polvere. Ciò include la limitazione della halo all'oggetto contenitore, ma anche le differenti funzioni di densità e di mappatura.
Un altro modo interessante per ottenere una distribuzione irregolare della polvere è quello di aggiungere turbolenza. L'effetto si ottiene usando la parola chiave `turbulence` seguita dalla quantità di turbolenza da usare (vedi **halo35.pov**).

```
box { -1, 1
pigment { colour rgbt <1, 1, 1, 1> }
halo {
dust
```

```
dust_type 1
box_mapping
linear
turbulence 1
colour_map {
[ 0 color rgbt <1, 1, 1, 1.0> ]
[ 1 color rgbt <1, 1, 1, 0.5> ]
}
samples 50
aa_level 3
aa_threshold 0.2
jitter 0.1
}
hollow
scale 5
}
```

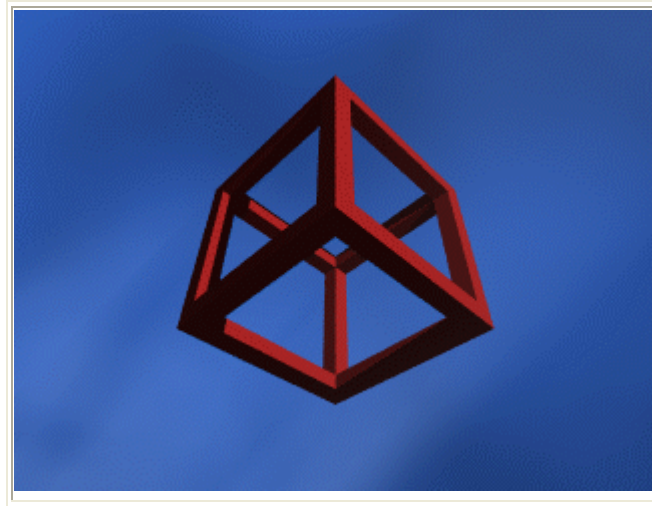



Fig. 155-Effetto della turbolenza

L'immagine che otteniamo è molto più interessante, a causa degli spostamenti nella densità delle particelle.

Dovremmo notare che usiamo una funzione di densità lineare (parola chiave `linear`) invece che la funzione costante. Ciò è necessario perché la funzione densità costante ha lo stesso valore in ogni punto. Aggiungere turbolenza non avrebbe effetto perché, comunque i punti venissero spostati, la densità avrebbe lo stesso valore. Aggiungere turbolenza ha senso solo se si usa una distribuzione di densità non costante. Il fatto che il valore della turbolenza è in effetti un vettore, può essere usato per creare effetti simili a cascate usando un alto valore di turbolenza in una sola direzione. (per esempio, `turbulence <0.2, 1, 0.2>`)

4.8.5.5.6 Usare Polvere Colorata

Se vogliamo creare una polvere colorata, possiamo facilmente ottenerla usando un colore diverso dal bianco nella mappatura del colore della halo. In questo caso, dovremo anche impostare il canale del filtro nella mappatura del colore ad un valore diverso da zero, per specificare la quantità di luce che viene filtrata dal colore della polvere. Useremo la mappatura del colore per ottenere una polvere rossa, parzialmente filtrante.

```
colour_map {
[ 0 color rgbft <1, 0, 0, 0.5, 1.0> ]
```

```
[ 1 color rgbft <1, 0, 0, 0.5, 0.7> ]
}
```

4.8.5.6 Tranelli degli Aloni

A causa della complessità della funzione e delle poche esperienze ancora fatte, ci sono ancora molte cose da scoprire riguardo agli aloni.
Alcuni dei problemi più comuni sono descritti nei paragrafi successivi per aiutarci ad evitarli.

4.8.5.6.1 Dove si Possono Usare gli Aloni

Come abbiamo già detto, un alone riempie completamente l'interno di un oggetto. Ricordandoci di questo, è ovvio che il seguente esempio non ha senso.

```
sphere { 0, 1
pigment {
checker
texture {
pigment { color Clear }
halo { ... }
}
texture {
pigment { color Red }
}
}
hollow
}
```

Cos'ha di sbagliato quest'esempio ? Semplicemente, si usa una halo per descrivere l'interno di un oggetto e non si può descrivere l'interno di un oggetto descrivendo come è la sua superficie. Non possiamo sapere a cosa assomiglierà l'interno della sfera. Sara completamente riempito dalla halo ? Ci saranno zone riempite dalla halo ed altre vuote ? Come saranno fatte ?
Non possiamo conoscere le proprietà dell'interno di un oggetto guardando la sua superficie. Non è possibile. Bisognerebbe sempre ricordarlo. Se l'esempio che abbiamo visto sopra fosse servito per creare una sfera riempita con una halo e coperta con un motivo a scacchiera che avesse parzialmente coperto la halo, avremmo usato la seguente sintassi :

```
sphere { 0, 1
pigment {
checker
texture {
pigment { color Clear }
}
texture {
pigment { color Red }
}
}
halo { ... }
hollow
}
```

Una halo viene sempre applicata ad un oggetto nel modo seguente :

```
OGGETTO {
```

```
texture {
pigment { ... }
normal { ... }
finish { ... }
halo { ... }
}
hollow
}
```

Non è possibile includere una halo in nessuna delle frasi `pigment{...}`, `color_map{...}`, `pigment_map{...}`, `texture_map{...}`, `material_map{...}` o qualunque altra. Non ci è impedito farlo, ma non otterremo ciò che vogliamo.
Possiamo usare gli aloni con texture stratificate nella misura in cui ci assicuriamo ghe gli aloni sono inclusi nell'ultimo strato (che deve comunque essere parzialmente trasparente per rendere visibile la halo).

4.8.5.6.2 Sovrapposizione degli Oggetti Contenitori

POV-Ray non è capace di gestire correttamente halo i cui oggetti contenitori si sovrappongono. Se creiamo due sfere parzialmente sovrapposte che contengono ciascuna una halo, non otterremo un risultato corretto dove le sfere si sovrappongono. L'effetto per la halo viene calcolato indipendentemente per ogni sfera ed i risultati vengono sommati.
Se vogliamo aggiungere halo diverse, dobbiamo posizionarle tutte all'interno di uno stesso contenitore per essere sicuri che siano calcolate correttamente (vedi anche "Halo Multiple").
Dovremmo anche notare che oggetti vicini, ma non sovrapposti, sono gestiti correttamente. Se poniamo un oggetto contenitore davanti ad un altro, le halo vengono renderizzate correttamente.

4.8.5.6.3 Aloni Attenuanti Multipli

Non è attualmente possibile utilizzare halo attenuanti multiple con mappature del colore diverse. La mappatura del colore dell'ultima verrà usata per tutte le altre halo che si trovano nel contenitore.

4.8.5.6.4 Aloni ed Oggetti Vuoti

Per renderizzare correttamente gli aloni, dobbiamo assicurarci che tutti gli oggetti entro cui si trova la camera siano vuoti. Ciò si ottiene usando la parola chiave `hollow`. (nel caso in cui la camera si trovi all'interno di un oggetto non vuoto, viene comunque visualizzato un messaggio di avvertimento, N.d.T.). Per una spiegazione dettagliata, vedi il paragrafo "Oggetti Vuoti ed Oggetti Solidi".

4.8.5.6.5 Ridimensionare un Contenitore

Se ridimensioniamo un contenitore, dovremmo ricordarci che ci sono grandi differenze a seconda di dove poniamo la parola chiave `scale`. Ridimensionare un oggetto prima della frase `halo{...}` ridimensionerà solo l'oggetto, non la halo. Ciò è utile se vogliamo evitare che la superficie dell'oggetto contenitore divenga visibile a causa dell'uso della turbolenza.
Come abbiamo visto nei paragrafi precedenti le particelle possono uscire dall'oggetto contenitore - diventando invisibili - a causa dell'uso della turbolenza. Ciò avviene solo per mappature di densità sferiche e cubiche dato che i campi di densità descritti dagli altri tipi di mappatura non hanno dimensioni finite.
Se usiamo la parola chiave `scale` dopo la frase `halo{...}`, sia la halo che l'oggetto contenitore verranno ridimensionati. Questo ci è utile per ridimensionare la halo a seconda delle nostre necessità.
La halo mantiene le sue caratteristiche indipendentemente dalle trasformazioni applicate all'oggetto

contenitore (dopo `halo{...}`), cioè trasparenza, colore e turbolenza della halo non vengono modificati.

4.8.5.6.6 Scegliere il Fattore di Campionamento

Normalmente, cominceremo con un basso tasso di campionamento e lo aumenteremo solo se si mostrano artificiosità dovute all'aliasing (che non riusciamo a rimuovere usando sovracampionamento e jittering). L'aspetto dell'alone è indipendente dal tasso di campionamento fino a che ci sono abbastanza campioni da dare una stima sufficientemente accurata del 'reale' aspetto della halo. Ciò significa che uno o due campioni sono appena il minimo necessario per determinare l'aspetto della halo. Aumentando il numero dei campioni, la halo tenderà rapidamente al suo aspetto 'reale'. In breve, l'aspetto della halo non cambierà col tasso di campionamento finché ci sono abbastanza campioni e non si presentano fenomeni di aliasing.

4.8.5.6.7 Usare la Turbolenza

Come abbiamo notato in uno dei paragrafi precedenti (vedi) la turbolenza non avrà alcun effetto se la applichiamo ad una halo con funzione densità costante. Non importa come o dove o di quanto spostiamo i punti della halo, dato che la densità è costante e quindi non dipende dalla posizione. Otterremo quindi lo stesso valore di densità per tutte le posizioni. (Aumentando il tempo di rendering di quel tanto necessario per il calcolo - in questo caso inutile, ma lo stesso costoso in termini di tempo - della turbolenza, N.d.T.)
Tutte le volte che aggiungiamo turbolenza ad una halo non dobbiamo usare la funzione di densità costante.

4.9 Lavorare con Texture Speciali

Molte delle colorazioni che abbiamo visto in altri paragrafi, facevano uso della frase `color_map` per unire insieme diversi colori. In dipendenza da come elenchiamo gli elementi della mappatura del colore, possiamo passare con gradualità da un colore all'altro, o possiamo invece avere una brusca variazione. In effetti, la mappatura dei colori è uno strumento molto potente per personalizzare le varie colorazioni e richiede un po' di esercizio per usarlo correttamente. E fin qui tutto bene, se ci basta utilizzare colori singoli. Ma se volessimo unire in motivi complessi interi pigmenti, o normali od intere texture ? A partire da POV-Ray 3, possiamo !
Per potere sperimentare alcune delle nuove opzioni, impostiamo un semplice file, nel quale inseriremo più tardi le texture di esempio. Per cominciare, mettiamoci i file include 'standard', una macchina fotografica ed una sorgente luminosa.

```
#include "colors.inc"
#include "textures.inc"

camera {
orthographic
up <0, 5, 0>
right <5, 0, 0>
location <0, 0, -25>
look_at <0, 0, 0>
}

light_source { <100, 100, -100> color White }
/*
DA QUI IN POI, INSERIRE GLI OGGETTI
RICHIESTI DA OGNI PARAGRAFO DEI
```

```
TUTORIAL SUCCESSIVI
*/
```

4.9.1 Mappatura di Pigmenti

Per iniziare con qualcosa di semplice, prendiamo in esame la mappatura di pigmenti, o `pigment_map`. Non dobbiamo confonderla con una mappatura di colore, poiché la mappatura di colore può comprendere elementi composti da singoli colori, mentre la mappatura di pigmenti può usare interi pigmenti. Per abituarci ad esse, iniziamo inserendo nella scena un semplice piano con una mappatura di pigmenti elementare. Nel seguente esempio, dichiareremo ognuno dei pigmenti che andremo ad usare prima di utilizzarlo. Ciò non è strettamente necessario (dato che per ogni elemento della mappatura, potremmo inserire la completa descrizione del pigmento) ma rende l'intero file più leggibile.

```
// scacchiera bianca e nera... un classico
#declare Pigment1 = pigment {
checker color Black color White
scale .1
}

// effetto 'anelli psichedelici'
#declare Pigment2 = pigment {
wood
color_map {
[ 0.0 Red ]
[ 0.3 Yellow ]
[ 0.6 Green ]
[ 1.0 Blue ]
}
}

plane { -z, 0
pigment {
gradient x
pigment_map {
[ 0.0 Pigment1 ]
[ 0.5 Pigment2 ]
[ 1.0 Pigment1 ]
}
}
}
```

Bene, ciò che abbiamo qui è molto semplice e probabilmente riconoscibile se abbiamo già lavorato sulle mappature di colore. Tutto ciò che abbiamo fatto è stato sostituire una mappa di pigmenti dove normalmente andrebbe una mappa di colori e come elementi della nostra mappatura, abbiamo messo i pigmenti dichiarati prima.

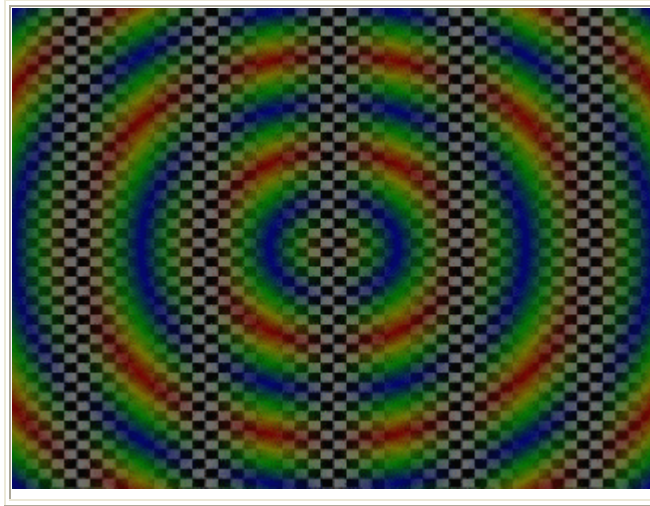



Fig. 156-Pigmenti sfumati l'uno nell'altro

Quando renderizziamo questo esempio, vediamo un motivo che sfuma alternativamente tra la classica scacchiera e gli anelli colorati. Dato che passiamo da `pigment1` a `pigment2` e viceversa, vediamo un chiaro sfumare dei due pattern l'uno nell'altro nelle zone di transizione. Potremmo ottenere un passaggio brusco altrettanto facilmente, modificando la mappa in questo modo :

```
pigment_map {
[ 0.0 Pigment1 ]
[ 0.5 Pigment1 ]
[ 0.5 Pigment2 ]
[ 1.0 Pigment2 ]
}
```

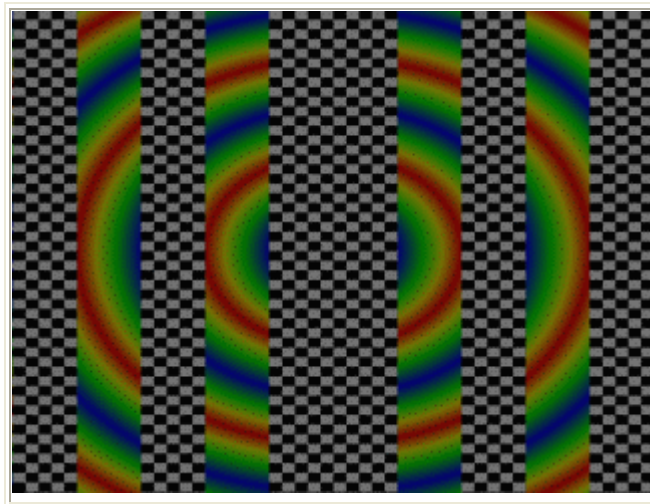



Fig. 157-Brusco stacco tra un pigmento e l'altro

Ma unire insieme pattern di pigmenti è solo l'inizio.

4.9.2 Mappatura delle Normali

Per il nostro prossimo esempio, sostituiamo il file nella scena con questo :

```
plane { -z, 0
pigment { White }
```

```
normal {
gradient x
normal_map {
0.0 bumps 1 scale .1]
1.0 ripples 1 scale .1]
}
}
}
```

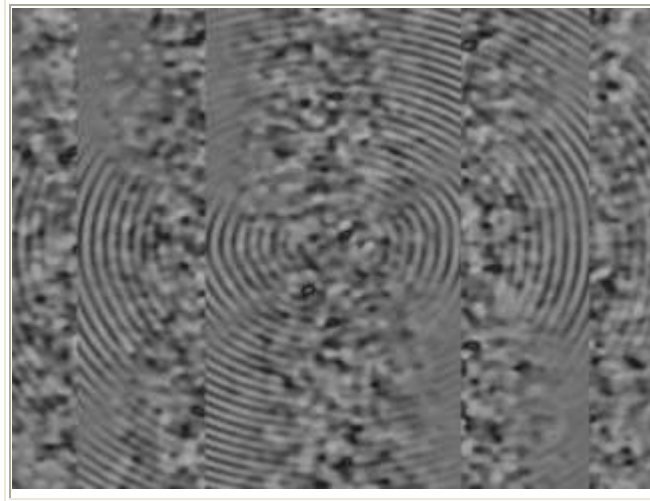



Fig. 158-Anche le normali possono essere mappate

Per prima cosa, abbiamo scelto un bianco a tinta unita per mostrare tutte le irregolarità della superficie al meglio. Poi, notiamo che la nostra mappa passa gradualmente da `bumps` a `ripples` tra 0 ed 1, ma poi essendo questo un gradiente, ritorna bruscamente a `bumps` all'inizio del passaggio successivo. Renderizziamo questo esempio e vediamo che ci sono abbastanza passaggi bruschi da poter vedere chiaramente dove i due motivi di normali si alternano, ma anche un esempio di come due normali si fondono uno nell'altro.
La sintassi è quella che ci saremmo aspettati. Abbiamo solo modificato il tipo di mappatura ed abbiamo aggiunto nel blocco delle normali i pattern appropriati. E' importante ricordare che in POV-Ray 3 tutti i pattern che funzionano con i pigmenti funzionano anche con le normali (e viceversa) e quindi potremmo avere un passaggio da legno (wood) a granito (granite) o qualunque altra cosa. Sperimentiamo per un po' e prendiamo confidenza con i differenti motivi.
Dopo avere visto come sono interessanti le normali di tipi diversi fuse insieme, potremmo volerle unire completamente, invece che sfumare dall'una all'altra. Bene, anche questo è possibile, ma staremmo facendo il passo più lungo della gamba. Questa funzione si chiama `average` (media) e ci ritorneremo sopra più avanti.

4.9.3 Mappatura delle Texture

Sappiamo come unire colori, pigmenti e normali e stiamo probabilmente pensando alle finiture. E intere texture ? Tutti e due possono essere trattati in un unico paragrafo. Mentre non c'è una mappatura delle finiture di per sé, ci sono le mappature di texture e quindi possiamo adattarle facilmente a svolgere il ruolo di mappature per la finitura utilizzando gli stessi pigmenti e/o normali in ciascuno dei componenti della mappa. Questo è un esempio. Eliminiamo i pigmenti che avevamo dichiarato prima ed aggiungiamo il testo seguente :

```
#declare Texture1 = texture {
pigment { Grey }
finish { reflection 1 }
```

```
}
#declare Texture2 = texture {
pigment { Grey }
finish { reflection 0 }
}
cylinder { <-2, 5, -2>, <-2, -5, -2>, 1
pigment { Blue }
}
plane { -z, 0
rotate y * 30
texture {
gradient y
texture_map {
[ 0.0 Texture1 ]
[ 0.4 Texture1 ]
[ 0.6 Texture2 ]
[ 1.0 Texture2 ]
}
scale 2
}
}
```

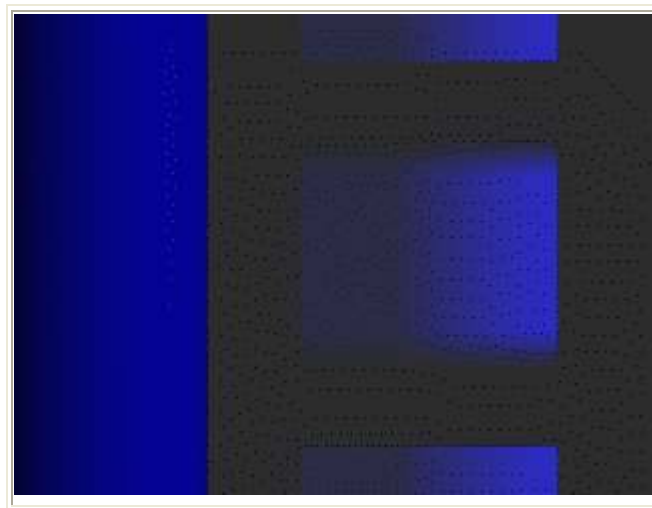



Fig. 159-Mappatura di texture

Ora, cosa abbiamo ottenuto ? Il piano cambia continuamente tra due texture, identiche in tutto tranne che nella finitura. Quando renderizziamo questa scena, il cilindro si riflette su parte del piano e su parte no. Con qualche piccolo adattamento, si potrebbe usare questo metodo con ogni pattern ed in ogni modo creativo, sia che vogliamo fornire finiture diverse a parti diverse dell'oggetto, come in questo caso, sia che vogliamo alternare texture completamente diverse. Ci si potrebbe chiedere : se esiste la mappatura delle texture, perché abbiamo bisogno di mappature dei pigmenti e delle normali ? Ottima domanda. La risposta è : velocità di calcolo. Se usiamo una mappatura di texture, per ogni punto intermedio tra due elementi della mappa, POV-Ray deve fare calcoli per ogni elemento della texture e quindi calcolare una media pesata per determinare il corretto colore di quel punto. Usando solo una mappatura di pigmenti (o di normali) si riduce il numero totale di operazioni e la nostra texture viene renderizzata un po' più velocemente. Come regola generale, si usano mappature di pigmenti e di normali dove si può e mappature di texture dove è richiesta una flessibilità superiore.

4.9.4 Lista di Texture

Se abbiamo seguito il tutorial corrispondente sui pigmenti semplici, sappiamo che ci sono tre motivi geometrici che vengono chiamati 'motivi a lista di colore', dato che invece di usare una mappatura di colore, questi motivi semplici ma utili usano una lista di colori che segue immediatamente la parola chiave che definisce il tipo di pattern. Stiamo parlando di `checker` (scacchiera), `hexagon` (esagoni) e, da POV-Ray 3, `brick` (mattoni). Naturalmente questi funzionano con pigmenti, normali ed intere texture, come gli altri pattern visti sopra. La sola differenza è che elenchiamo gli elementi che compongono il motivo in una lista (come faremmo con singoli colori), invece che distribuirli in una mappatura come sopra. Ecco un esempio. Cancelliamo il piano e tutte le dichiarazioni di pigmenti che avevamo fatto in precedenza ed aggiungiamo il testo seguente al nostro file base.

```
#declare Pigment1 = pigment {
hexagon
color Yellow color Green color Grey
scale .1
}

#declare Pigment2 = pigment {
checker
color Red color Blue
scale .1
}

#declare Pigment3 = pigment {
brick
color White color Black
rotate -90*x
scale .1
}

box { -5, 5
pigment {
hexagon
pigment {Pigment1}
pigment {Pigment2}
pigment {Pigment3}
rotate 90*x
}
}
```

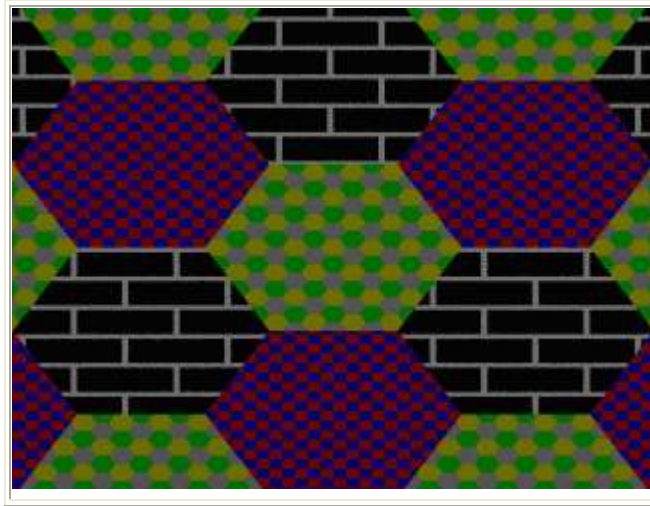



Fig. 160-Usare texture e motivi di colore

Iniziamo dichiarando ciascuno dei diversi pattern come un pigmento individuale. Poi usiamo il pattern ad esagoni come un pattern a lista di pigmenti, semplicemente fornendogli una lista di pigmenti anziché di colori come abbiamo fatto prima. Ci sono anche due frasi `rotate` in questo esempio, perché i 'mattoni' sono allineati lungo l'asse z, mentre gli esagoni sono allineati lungo l'asse y e noi vogliamo che tutti siano di fronte alla macchina fotografica, che abbiamo rivolto nella direzione z, in modo da poter vedere l'effetto dato dai pattern inclusi gli uni negli altri.
Naturalmente, i pattern a lista di colori funzionavano solo con i pigmenti, ma con POV-Ray 3, tutto quello che funzionava con i pigmenti può essere adattato alle normali o ad intere texture. Un paio di rapidi esempio potrebbero essere :

```
normal {
brick
normal { granite .1 }
normal { bumps 1 scale .1 }
}
```

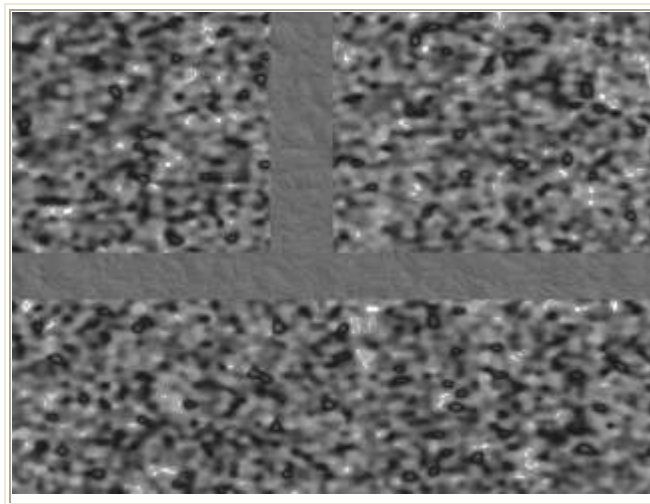



Fig. 161-Più normali nella stessa texture

oppure,

```
texture {
checker
```

```
texture { Gold_Metal }
texture { Silver_Metal }
}
```

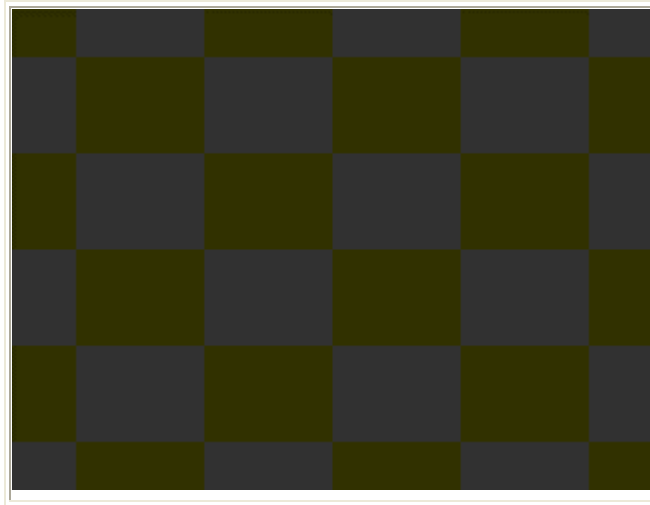



Fig. 162-Più texture

4.9.5 Che ne è Stato di Tiles ?

Nelle prime versioni di POV-Ray, c'era un pattern per le texture che si chiamava `tiles` (mattonelle) Utilizzando un semplice motivo a scacchiera, come abbiamo visto sopra, possiamo ottenere lo stesso risultato che si otteneva con `tiles`, che quindi è ormai obsoleto. E' tuttora supportato da POV-Ray 3 per compatibilità all'indietro, ma ora è tempo di abituarsi ad usare un motivo a scacchiera (e quindi la parola chiave `checker`) al suo posto.

4.9.6 Funzione Media

Ora le cose si fanno interessanti. Prima, abbiamo iniziato a vedere come pigmenti e normali possono essere sfumati gli uni negli altri quando li usiamo in mappature. Ma se volessimo avere una miscela di motivi su tutto l'oggetto ? Questo è il punto in cui una nuova funzione, chiamata `average`, viene utile. Average funziona con mappature di pigmenti, normali e texture, per quanto la sintassi vari leggermente e quando non ce lo aspettiamo, il cambiamento può confondere. Qui abbiamo un semplice esempio. Utilizziamo il solito file ed inseriamo i seguenti oggetti.

```
plane { -z, 0
pigment { White }
normal {
average
normal_map {
[ gradient x ]
[ gradient y ]
}
}
}
```

Ciò che abbiamo qui si spiega da solo quando lo renderizziamo.

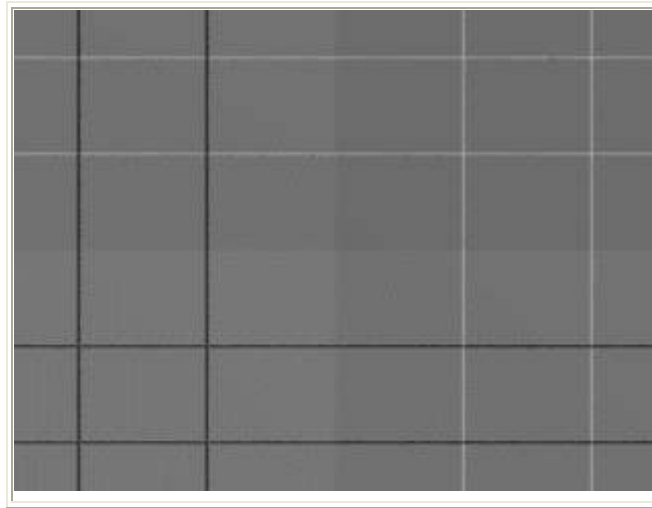



Fig. 163-"Scacchiera!"

Abbiamo unito un pattern di normali a bande verticali con uno a bande orizzontali creando due gradienti che si incrociano. In effetti, l'effetto di 'incrocio' è dato da una morbida fusione di `gradient x` con `gradient y` su tutto il nostro piano. Ora, dove sono le differenze ?
Vediamo che la nostra mappa di normali è cambiata dagli esempi che avevamo fatto prima. Il valore decimale a sinistra di ogni elemento della mappa è stato rimosso. Questo valore, normalmente serve a mappare ogni elemento della mappa sul pattern che abbiamo scelto, ma `average` è una unione su tutto il piano e quindi i numeri non servono. In effetti, includere i numeri può portare talvolta a risultati inaspettati, come una cattiva (o mancante) rappresentazione di alcuni degli elementi della mappa. Per assicurarci che otterremo l'unione di pattern che ci aspettiamo, lasciamo perdere il valore decimale.

4.9.7 Lavorare con Texture Stratificate

Con la moltitudine di colori, pattern e opzioni per creare texture complesse in POV-Ray, possiamo facilmente diventare esperti nel mescolare ed armeggiare le texture giuste da applicare alle nostre creazioni. Ma mentre lavoriamo, prima o poi si deve arrivare a *quella* texture speciale. Quella texture che assomiglia un po' a del legno, solo verniciato, e con un po' di venature gialle e qualche passata di grigio in verticale, che fa sembrare che qualcuno abbia cominciato a verniciarci sopra e poi si sia fermato, lasciando solo parte del legno visibile attraverso la vernice.
Solo che... ed ora ? Come facciamo a far stare tutta quella roba in una sola texture ? Non ci sono pattern che possono fare tutte quelle cose. Prima di entrare nel panico ed adottare una bitmap da mappare sull'oggetto, c'è al minimo un'altra possibilità : le *texture stratificate*.
Le texture stratificate consistono nello specificare una serie di texture, una dopo l'altra, tutte associate con lo stesso oggetto. Tutte le texture che noi specifichiamo verranno applicate, l'una dopo l'altra, sullo stesso oggetto, dal basso verso l'alto (o meglio, dagli strati più interni a quelli più esterni). E' molto importante notare che dobbiamo avere una qualche trasparenza, di tipo filtrante (filter) o non filtrante (transmit) nei pigmenti delle texture degli strati superiori, oppure gli strati sottostanti saranno invisibili. Non riceveremmo messaggi di errore, perché ciò è possibile, semplicemente, non ha senso. E' come spendere delle ore per tracciare un elaborato disegno su di un muro e poi verniciare tutto a tinta unita. Creiamo un oggetto molto semplice con una texture stratificata e vediamo cosa succede : creiamo un file denominato **laytex.pov** e vi inseriamo il seguente testo :

```
#include "colors.inc"
#include "textures.inc"
```

```
camera {
location <0, 5, -30>
look_at <0, 0, 0>
}

light_source { <-20, 30, -50> color White }

plane { y, 0 pigment { checker color Green color Yellow } }

background { rgb <.7, .7, 1> }

box { <-10, 0, -10>, <10, 10, 10>
texture {
Silver_Metal // un oggetto di metallo ...
normal { // ... che ha preso un colpo
dents 2
scale 1.5
}
} // (fine della texture base)

texture { // ... ha qualche macchia di ruggine...
pigment {
granite
color_map {
[0.0 rgb <.2, 0, 0> ]
[0.2 color Brown ]
[0.2 rgbt <1, 1, 1, 1> ]
[1.0 rgbt <1, 1, 1, 1> ]
}
frequency 16
}
} // (fine della texture a macchie di ruggine)

texture { // ... e qualche segno di fuliggine
pigment {
bozo
color_map {
[0.0 color Black ]
[0.2 color rgbt <0, 0, 0, .5> ]
[0.4 color rgbt <.5, .5, .5, .5> ]
[0.5 color rgbt <1, 1, 1, 1> ]
[1.0 color rgbt <1, 1, 1, 1> ]
}
scale 3
}
} // (fine della texture fuliggine)

} // (fine della frase box )
```

Qui le cose si fanno complicate, allora per rendere il file più leggibile, abbiamo aggiunto commenti che indicano cosa stiamo facendo e dove terminano le varie parti delle texture (in modo da non perdersi in tutte quelle parentesi !).

Per cominciare, creiamo un semplice parallelepipedo sul classico pavimento a scacchiera e forniamo al cielo un colore blu pallido. Ed ora, la parte divertente. Abbiamo usato per il parallelepipedo la texture `silver_metal` che è dichiarata nel file **textures.inc**. Ovviamente, è meglio andare a leggere il file per vedere come la texture è stata effettivamente creata. Per dare al metallo un aspetto consunto, abbiamo aggiunto il pattern di normali `dents` che crea l'illusione che sulla superficie siano stati battuti dei colpi, come se la nostra misteriosa scatola di metallo fosse stata maltrattata per un bel po'. Le macchie di ruggine non sono altro che un pattern `granite` molto fine che sfuma dal rosso scuro al marrone e raggiunge poi la piena trasparenza nella maggior parte della mappatura del colore. E' vero, probabilmente avremmo potuto inventarci un pattern più realistico, usando delle mappature di pigmenti per raggruppare le macchie di ruggine, ma le mappature di pigmenti sono materia di un altro paragrafo del tutorial quindi, per adesso accontentiamoci.
Infine, abbiamo aggiunto una terza texture all'insieme, la texture `bozo` che gradualmente sfuma da zone scure a zone di grigio medio semitrasparente ed infine a zone del tutto trasparenti nella seconda metà della sua mappatura del colore. Questo ci dà l'idea di bruciature fuligginose che ancora di più tormentano la superficie della nostra scatola di metallo. Il risultato finale lascia la superficie della nostra scatola di metallo con un aspetto veramente consunto, utilizzando delle texture a strato multiplo per produrre un effetto che nessuna texture semplice avrebbe potuto produrre.

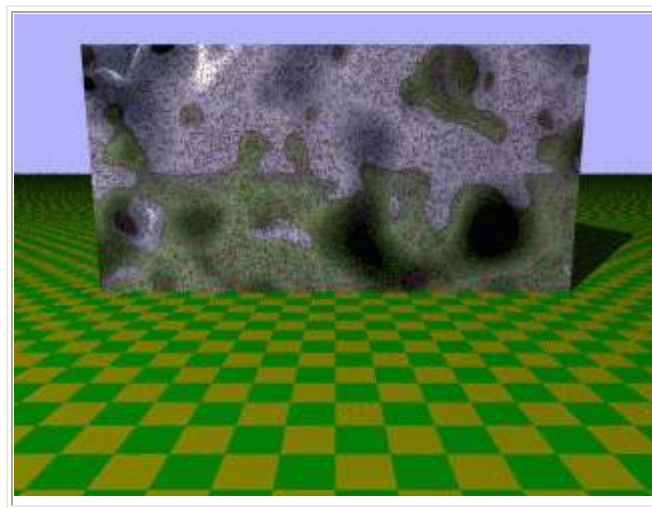



Fig.164-Effetto finale

4.9.7.1 Dichiarare Texture Stratificate

Nel caso in cui noi volessimo utilizzare una texture stratificata su alcuni oggetti della nostra scena, è perfettamente possibile dichiarare una texture stratificata. Non ripeteremo la texture che abbiamo visto nel paragrafo precedente, ma il risultato finale dovrebbe essere qualcosa di questo genere.

```
#declare Metallo_Consumato =
texture { /* inserisci la texture di base qui... */ }
texture { /* le macchie di ruggine qui... */ }
texture { /* e naturalmente, le bruciature fuligginose qui */ }
```

POV-Ray non ha problemi nell'individuare il punto in cui terminano le texture di una dichiarazione, dato che le texture si susseguono l'una dopo l'altra senza oggetti o direttive nel mezzo. La texture stratificata che viene dichiarata sarà assunta continuare finché non si troverà qualcosa di diverso da una frase `texture{...}`. Quindi in questo modo un qualunque numero di strati può essere aggiunto alla dichiarazione di texture. Un'ultima cosa circa le texture stratificate. Qualunque texture

stratificata usiamo, dichiarata o no, non dobbiamo dimenticare la frase `texture{...}` che la circonda. In texture convenzionali singole, una normale scorciatoia è quella di indicare solo il pigmento, o solo il pigmento e la finitura, o solo le normali, o qualunque altra cosa, senza racchiudere il tutto in una frase `texture{...}`. Questa scorciatoia non si estende alle texture stratificate. Noi, con POV-Ray possiamo stratificare intere texture ma non parti individuali di texture. Per esempio,

```
#declare Falsa_Texture =
texture { /* inserisci la texture base qui... */ }
pigment { Red filter .5 }
normal { bumps 1 }
```

non funzionerebbe. I pigmenti e le normali sono lì senza essere parte di una qualunque particolare texture. Entro un oggetto, con una singola texture, possiamo fare questo genere di cose, ma con texture stratificate otterremmo solo un messaggio di errore, sia che questa si trovi all'interno di un oggetto, sia che si trovi all'interno di una dichiarazione.

4.9.7.2 Un Altro Esempio di Texture Stratificata

Per spiegare ulteriormente come funzionano le texture stratificate, viene descritto in dettaglio un altro esempio. Diciamo di voler creare una tovaglia da usare per una scena di un picnic. Dato che una semplice tovaglia a scacchi rossi e bianchi ha un aspetto troppo nuovo, troppo piatto e troppo somigliante ad un pavimento, usiamo una texture stratificata per creare questo panno. Ora, creiamo una scena che contiene quattro cubi. Il primo cubo ha la semplice texture rossa e bianca con la quale abbiamo iniziato la nostra scena del picnic. Il secondo ha uno strato che ha la funzione di sfumare il colore della nostra tovaglia. Il terzo aggiunge qualche macchia di vino e l'ultimo aggiunge anche qualche increspatura. Non è in effetti un altro strato vero e proprio, ma dobbiamo stare attenti a quando e come le normali hanno un effetto nelle texture stratificate. Iniziamo, per prima cosa, a piazzare una macchina fotografica, delle luci ed il primo parallelepipedo. A questo stadio, la texture è semplicemente non stratificata (vedi **layered1.pov**).

```
#include "colors.inc"

camera {
location <0, 0, -6>
look_at <0, 0, 0>
}

light_source { <-20, 30, -100> color White }
light_source { <10, 30, -10> color White }
light_source { <0, 30, 10> color White }

#declare PLAIN_TEXTURE =
// scacchi bianchi e rossi
texture {
pigment {
checker
color rgb<1.000, 0.000, 0.000>
color rgb<1.000, 1.000, 1.000>
scale <0.2500, 0.2500, 0.2500>
}
}
```

```
// parallelepipedo a scacchi bianchi e rossi

box { <-1, -1, -1>, <1, 1, 1>
texture {
PLAIN_TEXTURE
}
translate <-1.5, 1.2, 0>
}
```

Renderizziamo questa scena.

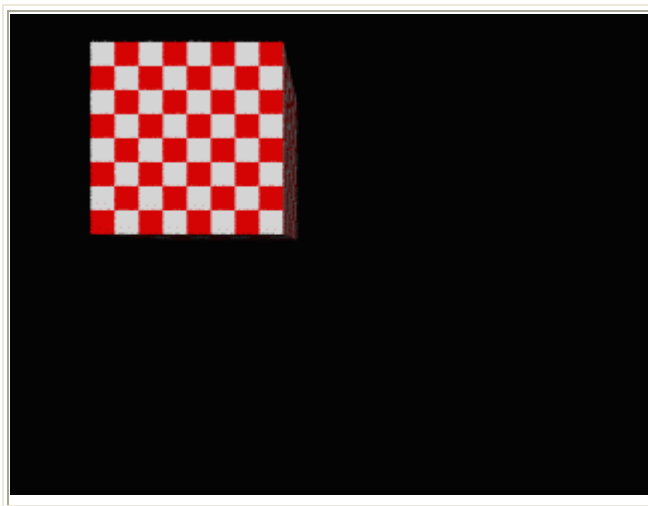



Fig. 165-Primo campione

Non è particolarmente interessante, vero ? Questo è il motivo per cui useremo delle texture stratificate. Per prima cosa, aggiungiamo uno strato di due grigi diversi, parzialmente trasparenti e li disponiamo come avevamo disposto i colori rosso e bianco, ma aggiungiamo loro anche un po' di turbolenza per rendere la sfumatura più realistica. Aggiungiamo la seguente scatola alla scena precedente e renderizziamo di nuovo (vedi file **layered2.pov**)

```
#declare FADED_TEXTURE =
// scacchi rossi e bianchi
texture {
pigment {
checker
color rgb<0.920, 0.000, 0.000>
color rgb<1.000, 1.000, 1.000>
scale <0.2500, 0.2500, 0.2500>
}
}
// grigi per sfumare il rosso e il bianco
texture {
pigment {
checker
color rgbf<0.632, 0.612, 0.688, 0.698>
color rgbf<0.420, 0.459, 0.520, 0.953>
turbulence 0.500
scale <0.2500, 0.2500, 0.2500>
}
```

```
}

// parallelepipedo a scacchi rossi e bianchi sfumati

box { <-1, -1, -1>, <1, 1, 1>
texture {
FADED_TEXTURE
}
translate <1.5, 1.2, 0>
}
```

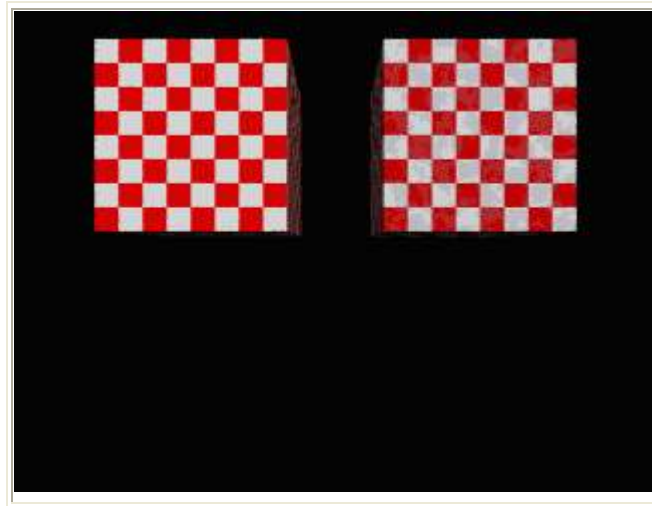



Fig. 166-Secondo

Nonostante che sia una differenza piuttosto sottile, notiamo che la tovaglia non sembra più così nuova. Dato che vogliamo inserire poi una bottiglia di vino nella scena del picnic, abbiamo pensato che potrebbe essere un tocco simpatico l'aggiungere una macchia di vino o due. Questo esempio potrebbe essere ottenuto aggiungendo un blob appiattito, ma darebbe l'effetto di vino rovesciato su un tavolo , non di una macchia. Quindi, dobbiamo aggiungere un altro strato. Di nuovo, aggiungiamo un altro parallelepipedo e renderizziamo nuovamente.

```
#declare STAINED_TEXTURE =
// scacchi rossi e bianchi
texture {
pigment {
checker
color rgb<0.920, 0.000, 0.000>
color rgb<1.000, 1.000, 1.000>
scale <0.2500, 0.2500, 0.2500>
}
}
// grigi per sfumare gli scacchi
texture {
pigment {
checker
color rgbf<0.634, 0.612, 0.688, 0.698>
color rgbf<0.421, 0.463, 0.518, 0.953>
turbulence 0.500
scale <0.2500, 0.2500, 0.2500>
}
```

```
}
// macchie di vino
texture {
pigment {
spotted
color_map {
[ 0.000 color rgb<0.483, 0.165, 0.165> ]
[ 0.329 color rgbf<1.000, 1.000, 1.000, 1.000> ]
[ 0.734 color rgbf<1.000, 1.000, 1.000, 1.000> ]
[ 1.000 color rgb<0.483, 0.165, 0.165> ]
}
turbulence 0.500
frequency 1.500
}
}

// parallelepipedo macchiato

box { <-1, -1, -1>, <1, 1, 1>
texture {
STAINED_TEXTURE
}
translate <-1.5, -1.2, 0>
}
```

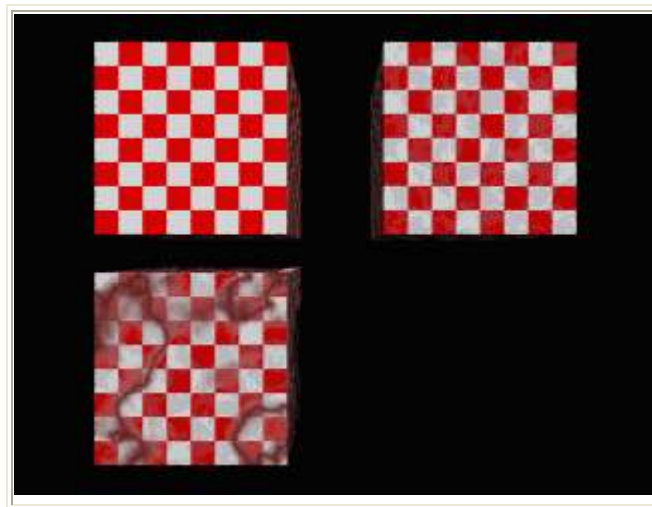



Fig. 167-Terzo campione

Ed ora abbiamo una texture per una tovaglia con una sua personalità. Un ultimo tocco che possiamo aggiungere alla tovaglia è qualche spiegazzatura, per dare alla tovaglia un'apparenza un po' 'vissuta'. Questo in effetti non è un altro strato vero e proprio, ma quando lavoriamo con le texture stratificate, dobbiamo fare attenzione che le modifiche alle normali alla superficie si trovino sullo strato più esterno delle texture. I cambiamenti alle normali degli strati interni non hanno alcun effetto sulla texture finale, indipendentemente dalla trasparenza degli strati soprastanti. Aggiungiamo quest'ultimo parallelepipedo al file e renderizziamo nuovamente (vedi il file **layered4.pov**).

```
#declare WRINKLED_TEXTURE =
// scacchi rossi e bianchi
texture {
```

```
pigment {
checker
color rgb<0.920, 0.000, 0.000>
color rgb<1.000, 1.000, 1.000>
scale <0.2500, 0.2500, 0.2500>
}
}
// grigi per sfumare gli scacchi
texture {
pigment {
checker
color rgbf<0.632, 0.612, 0.688, 0.698>
color rgbf<0.420, 0.459, 0.520, 0.953>
turbulence 0.500
scale <0.2500, 0.2500, 0.2500>
}
}
// le macchie di vino
texture {
pigment {
spotted
color_map {
[ 0.000 color rgb<0.483, 0.165, 0.165> ]
[ 0.329 color rgbf<1.000, 1.000, 1.000, 1.000> ]
[ 0.734 color rgbf<1.000, 1.000, 1.000, 1.000> ]
[ 1.000 color rgb<0.483, 0.165, 0.165> ]
}
turbulence 0.500
frequency 1.500
}
normal {// qui vengono aggiunte le spiegazzature
wrinkles 5.0000
}
}

// parallelepipedo 'spiegazzato'

box { <-1, -1, -1>, <1, 1, 1>
texture {
WRINKLED_TEXTURE
}
translate <1.5, -1.2, 0>
}
```

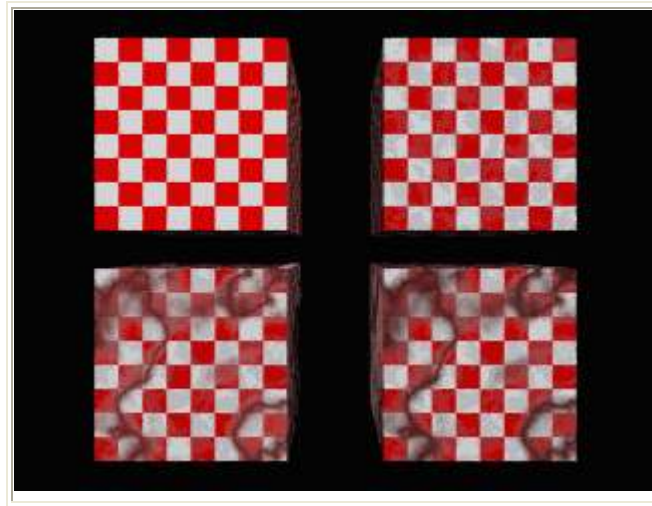



Fig. 168-Quarto campione

Bene, questa potrebbe non essere la tovaglia che vorremmo invitare ad un picnic, ma se confrontiamo l'ultimo parallelepipedo con il primo, vediamo quanta profondità, dimensione e personalità posiamo aggiungere ad un oggetto utilizzando texture appropriate.
Un ultima nota : quanto vale per le normali nelle texture stratificate, non vale per le finiture. Se una texture di uno strato inferiore contiene una finitura, per esempio a specchio, questa è visibile in tutti i punti in cui la trasparenza degli strati soprastanti lo consente.

4.9.8 Quando Fallisce Tutto il Resto : Mappatura dei Materiali

Abbiamo a nostra disposizione alcuni strumenti molto potenti per creare texture, ma se volessimo disporre texture complesse in maniera più libera ? Bene, così come la mappatura di un'immagine funziona per i pigmenti e la mappatura delle normali funziona per le normali, possiamo mappare intere texture usando una mappatura dei materiali.
Come per le mappature di immagini e di normali, abbiamo bisogno di un'immagine di partenza in un formato che possa essere letto da POV-Ray, da usare come mappa per determinare dove le singole texture andranno, ma in questo caso abbiamo bisogno di specificare quale texture sarà associata ad un determinato numero d'ordine nella tavolozza dei colori dell'immagine. Per fare quest'immagine possiamo usare un programma di disegno che ci permetta di selezionare i colori a partire dal numero che hanno all'interno della tavolozza (il vero colore è irrilevante, dal momento che si tratta solo di una mappa che dice a POV-Ray quale texture sarà usata in quella posizione).
Ora, se abbiamo il pacchetto completo di POV-Ray, possiamo trovare nella directory **include** un'immagine chiamata **povmap.gif** che è un'immagine bitmap che usa solo i primi quattro colori della tavolozza per creare una cornice quadrata con le parole *Persistance Of Vision* all'interno.

Fig. 169-Immagine da usare come mappa

Quest'immagine andrà già bene per una mappatura di esempio per la dimostrazione che segue. Usando gli stessi file .inc, la stessa macchina fotografica e la stessa luce degli esempi precedenti, inseriamo l'oggetto che segue.

```
#include "textures.inc"
plane { -z, 0
texture {
material_map {
gif "povmap.gif"
interpolate 2
once
texture { PinkAlabaster } // bordo interno
texture { pigment { DMFDarkOak } } // bordo esterno
texture { Gold_Metal } // lettere
texture { Chrome_Metal } // sfondo
}
translate <-0.5, -0.5, 0>
scale 5
}
}
```

Fig. 170-Prima prova

La posizione della sorgente luminosa e la mancanza di oggetti che si possano riflettere non permettono di mostrare al loro meglio queste texture. Ma almeno possiamo vedere come funziona il procedimento. Le texture sono state semplicemente disposte in accordo alla posizione dei pixel il cui colore corrisponde ad un particolare numero d'ordine della tavolozza. Usando la parola chiave `once` (per evitare che le texture risultino affiancate) e traslando e scalando la nostra mappa affinché questa possa rientrare all'interno dell'inquadratura che avevamo usato, arriviamo a vedere l'intera scena. Naturalmente ciò vale solo per immagini che sfruttano la tavolozza dei colori, come GIF ed alcuni tipi di PNG. Le mappature di materiali possono anche usare formati che non usano la tavolozza di colori, come i file TGA che costituiscono l'output di POV-Ray. Ciò porta ad un'interessante conseguenza : possiamo usare POV-Ray per produrre mappe per POV-Ray stesso ! Prima di concludere con alcune delle limitazioni di queste texture speciali, facciamo un'altra cosa con le mappature dei materiali per far vedere come POV-Ray può prodursi le mappe. Per cominciare, se si sta usando un'immagine che non fa uso della tavolozza, POV-Ray guarda al componente rosso del colore del pixel (che può avere un valore da 0 a 255) per determinare quale texture della lista usare. Quindi per creare una mappa abbiamo bisogno di controllare con esattezza

quale sarà il valore di rosso di un determinato pixel. Possiamo fare questo controllo così :

1) Usando una frase `rgb` per scegliere il nostro colore. Ad esempio, `rgb<x/255,0,0>`, dove "x" è il valore di rosso che vogliamo assegnare a quel pigmento e poi...
2) Evitando di usare sorgenti luminose ed applicando una finitura `finish{ambient 1}` a tutti gli oggetti, per assicurarsi che le luci e le ombre non interferiscano.

Confusi ? Bene, questo è un esempio che genererà una mappa molto simile a **povmap.gif**, che abbiamo usato prima, tranne per il formato che sarà **.TGA**. Notiamo che sono specificati anche i componenti di colore blu e verde. POV-Ray li ignorerà nell'immagine finale, ma sono specificati per noi umani, poiché i nostri occhi non possono distinguere la differenza tra variazioni nel rosso tra 0 e 4/255. Senza le variazioni di blu e verde la nostra mappa ci apparirebbe come un'immagine completamente nera. Questo può essere un modo geniale di mandare messaggi segreti usando POV-Ray (inserirli in una mappa di materiali da decodificare), ma è inutile se vogliamo vedere come è la nostra mappa.
Renderizziamo il seguente file e rinominiamo l'immagine risultante **povmap.tga**

```
camera {
orthographic
up <0, 5, 0>
right <5, 0, 0>
location <0, 0, -25>
look_at <0, 0, 0>
}

plane { -z, 0
pigment { rgb <1/255, 0, 0.5> }
finish { ambient 1 }
}

box { <-2.3, -1.8, -0.2>, <2.3, 1.8, -0.2>
pigment { rgb <0/255, 0, 1> }
finish { ambient 1 }
}

box { <-1.95, -1.3, -0.4>, <1.95, 1.3, -0.3>
pigment { rgb <2/255, 0.5, 0.5> }
finish { ambient 1 }
}

text { ttf "crystal.ttf", "The vision", 0.1, 0
scale <0.7, 1, 1>
translate <-1.8, 0.25, -0.5>
pigment { rgb <3/255, 1, 1> }
finish { ambient 1 }
}

text { ttf "crystal.ttf", "Persists!", 0.1, 0
scale <0.7, 1, 1>
translate <-1.5, -1, -0.5>
pigment { rgb <3/255, 1, 1> }
```

```
finish { ambient 1 }
}
```

Fig. 171-Un'altra immagine da usare come mappa

Tutto quello che dobbiamo fare è modificare il nostro ultimo esempio di mappatura di materiali cambiando la mappatura del materiale da GIF a TGA e cambiando il nome del file. Quando faremo il rendering usando la nuova mappa il risultato sarà molto simile a quello che abbiamo ottenuto con l'immagine GIF che teneva conto dei colori della tavolozza, tranne che non abbiamo dovuto usare un programma esterno per generare l'immagine : ha fatto tutto POV-Ray !

Fig. 172

4.9.9 Limiti delle Texture Speciali

Ci sono un paio di limitazioni per tutte le texture speciali che abbiamo visto. Per prima cosa, se abbiamo usato le direttive di default per impostare le texture di default per tutti gli oggetti nella nostra scena, allora POV-Ray non accetterà nessuna delle texture speciali discusse qui. Questo è un problema decisamente minore, dal momento che possiamo sempre dichiarare una texture ed applicarla individualmente a tutti gli oggetti. Non ci impedisce insomma di fare cose che non potremmo fare in altro modo.

L'altra limitazione è più restrittiva, ma come vedremo tra breve, possiamo aggirarla piuttosto facilmente. Abbiamo lavorato con texture stratificate ed abbiamo già visto come possiamo accumulare texture una sopra l'altra (fino a che ogni strato contiene della trasparenza). Questa tecnica molto utile ha un problema nell'incorporare le texture speciali che abbiamo appena visto

come strati. Ma c'è una risposta !
Per esempio, diciamo di avere una texture stratificata chiamata `Speckled_Metal`, che determini una superficie argentea, con piccoli punti di ruggine. Possiamo decidere, per maggiore realismo, di creare macchie di ruggine più grandi, disposte casualmente sulla superficie. L'approccio più ovvio è quello di creare una texture speciale, con trasparenza, da usare come strato superiore, ma naturalmente, come abbiamo visto, non saremo in grado di usare quel motivo di texture come strato. Otterremmo solamente un messaggio di errore. La soluzione è di capovolgere il problema e far diventare la nostra texture stratificata parte della texture stessa, come nell'esempio :

```
// Questa parte dichiara un pigmento da
// usare nella texture 'macchie di ruggine'
#declare Rusty = pigment {
granite
color_map {
[ 0 rgb <0.2, 0, 0> ]
[ 1 Brown ]
}
frequency 20
}

// E questa parte lo applica.
// Nota che la nostra texture stratificata originale
// "Speckled_Metal" è ora parte della mappa
#declare Rust_Patches = texture {
bozo
texture_map {
[ 0.0 pigment {Rusty} ]
[ 0.75 Speckled_Metal ]
[ 1.0 Speckled_Metal ]
}
}
```

L'effetto che otteniamo è in definitiva come se avessimo messo lo strato 'macchie di ruggine' sopra lo strato 'Speckled_Metal'.
Con l'intera gamma di pattern, pigmenti, normali, finiture, texture speciali e stratificate, ora possiamo praticamente fare tutto. C'è un numero quasi infinito di combinazioni che aspettano solo di essere create !

4.10 Usare Effetti Atmosferici

POV-Ray offre una quantità di effetti atmosferici, cioè di funzioni che influiscono sullo sfondo della scena o sull'aria di cui tutto è circondato. E' semplice assegnare un singolo colore od un complesso motivo di colori ad una virtuale sfera celeste. Puoi creare qualunque cosa, da un cielo estivo senza nuvole, ad un cielo tempestoso. Si possono creare facilmente anche cieli stellati.
Si possono usare diversi tipi di nebbia per creare scene nebbiose, strati multipli di differenti colori di nebbia possono aggiungere un tocco di mistero alla tua scena.
Un effetto più realistico può essere creato usando un'atmosfera, una nebbia continua che interagisca con le luci. I raggi di luce diventeranno visibili e gli oggetti proietteranno ombre nella nebbia. Infine, si possono aggiungere arcobaleni.

4.10.1 Lo Sfondo

La funzione `background` (sfondo) è usata per assegnare un colore a tutti i raggi che non colpiscono nessun oggetto.

```
#include "colors.inc"
camera {
location <0, 0, -10>
look_at <0, 0, 0>
}
light_source { <10, 10, -10> White }
background { color rgb <0.2, 0.2, 0.3> }
sphere { 0, 1
pigment { color rgb <0.8, 0.5, 0.2> }
}
```

Il colore dello sfondo sarà visibile se verrà usata una sfera celeste e se alcune parti trasparenti rimarranno dopo che tutti gli strati di pigmento della sfera celeste saranno stati assegnati.

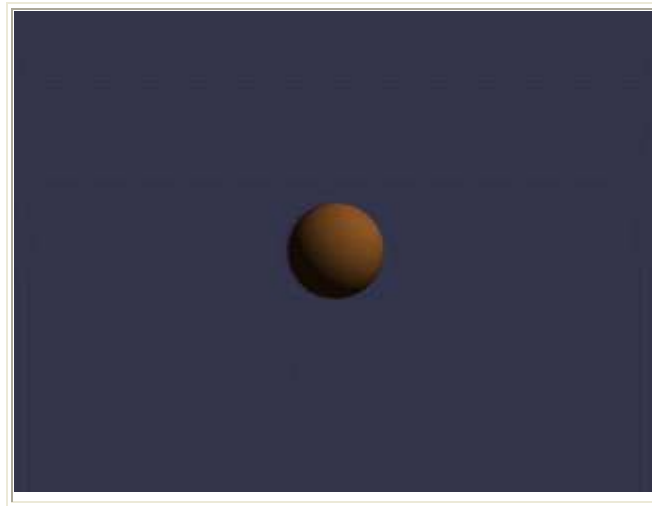



Fig.173-L'uso dello sfondo

4.10.2 La Sky Sphere (Sfera Celeste)

La sfera celeste può essere usata per creare facilmente un cielo nuvoloso, una notte stellata, o qualunque tipo di cielo tu abbia in mente. Negli esempi che seguiranno, abbiamo iniziato con una sfera celeste molto semplice, che è resa sempre più complessa aggiungendole nuove funzioni.

4.10.2.1 Creare un Cielo Usando un Gradiente di Colore

Oltre alla sfera celeste che utilizza un solo colore, effetto che si può ottenere usando lo sfondo, la sfera celeste più semplice è un gradiente di colore. Puoi avere notato che il colore del cielo varia con l'angolo formato con la superficie della terra. Se guardi in alto il cielo normalmente ha un blu molto più profondo di quello che ha all'orizzonte. Vogliamo modellare questo effetto usando la sfera celeste come nell'esempio che segue (vedi **skysph1.pov**).

```
#include "colors.inc"
camera {
location <0, 1, -4>
look_at <0, 2, 0>
angle 80
}
```

```
light_source { <10, 10, -10> White }
sphere { 2*y, 1
pigment { color rgb <1, 1, 1> }
finish { ambient 0.2 diffuse 0 reflection 0.6 }
}
sky_sphere {
pigment {
gradient y
color_map {
[0 color Red]
[1 color Blue]
}
scale 2
translate -1
}
}
```

Fig. 174-Usare i gradienti

La parte interessante è la frase `sky_sphere{...}`. Contiene un pigmento che descrive l'aspetto del cielo. Dato che viene usato un vettore direzione per calcolare i colori del pigmento dobbiamo usare un `gradient y`.
Le trasformazioni `scale` e `translate` sono usate per mappare i punti ottenuti dal vettore direzione nell'intervallo corretto. Senza queste trasformazioni il gradiente sarebbe ripetuto due volte sulla sfera celeste. L'istruzione `scale` è usata per evitare la ripetizione del gradiente e l'istruzione `translate -1` sposta la componente 0 della `color_map` sul fondo della sfera (il punto della sfera celeste che vedi guardando in giù). Dopo questa trasformazione il colore alla posizione zero nella mappatura sarà sul fondo della sfera celeste, cioè sotto di noi e il colore alla posizione 1 sarà in cima cioè sopra di noi.
I colori di tutte le altre posizioni sono interpolati tra queste due come si può vedere nell'immagine risultante.
Se vuoi far iniziare uno dei colori ad un particolare angolo dovrai prima convertire l'angolo ad un valore da inserire nella mappatura dei colori. Ciò si ottiene usando la formula :

```
valore =(1-cos(angolo))/2
```

dove l'angolo è misurato rispetto alla normale negativa della superficie della terra, cioè la normale alla superficie che punta verso il centro della terra. Un angolo di 0 gradi descrive il punto sotto di

noi, mentre un angolo di 180 gradi rappresenta lo zenit.
In POV-Ray devi prima convertire il valore in gradi a radianti come si mostra nel seguente esempio.

```
sky_sphere {
pigment {
gradient y
color_map {
[(1-cos(radians( 30)))/2 color Red]
[(1-cos(radians(120)))/2 color Blue]
}
scale 2
translate -1
}
}
```

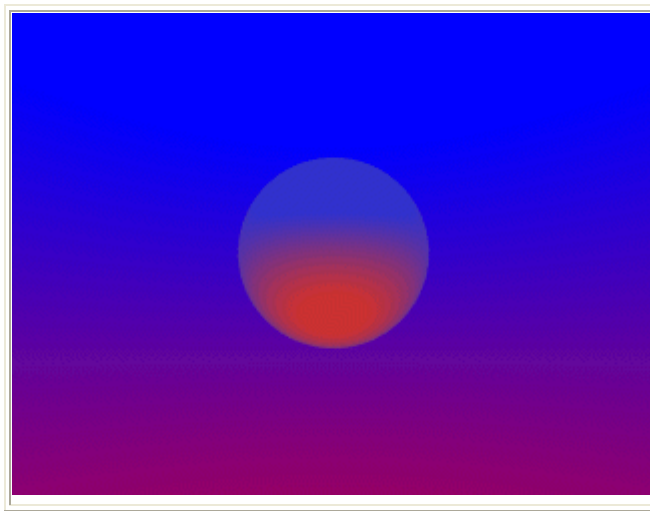



Fig. 175-Gradiente

Questa scena usa un gradiente di colore che inizia col rosso a 30 gradi e sfuma nel blu a 120 gradi. Sotto i 30 gradi è tutto rosso mentre sopra i 120 è tutto blu.

4.10.2.2 Aggiungere il Sole

Nell'esempio seguente creeremo un cielo con un sole rosso circondato da un alone rosso che sfuma nel blu della notte. Otterremo questo usando solo la funzione `sky_sphere`.
La `sky_sphere` che usiamo è mostrata sotto. Abbiamo aggiunto anche un piano per aumentare il realismo (vedi **skysph2.pov**).

```
sky_sphere {
pigment {
gradient y
color_map {
[0.000 0.002 color rgb <1.0, 0.2, 0.0>
color rgb <1.0, 0.2, 0.0>]
[0.002 0.200 color rgb <0.8, 0.1, 0.0>
color rgb <0.2, 0.2, 0.3>]
}
scale 2
translate -1
}
```

```
rotate -135*x
}

plane { y, 0
pigment { color Green }
finish { ambient .3 diffuse .7 }
}
```

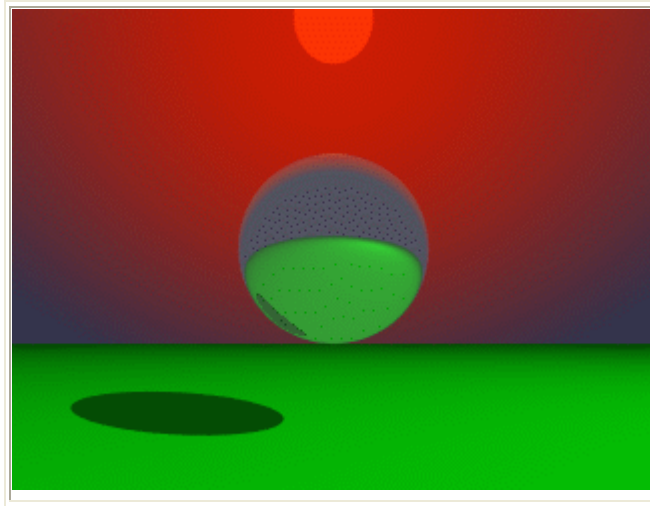



Fig. 176-Il sole

Il gradiente e le trasformazioni del pigmento sono le stesse che abbiamo visto nel paragrafo precedente. La mappa dei colori consiste di tre componenti : un rosso brillante e tendente al giallo usato per il sole, un rosso più scuro per l'alone e blu scuro per il cielo notturno. Il colore del sole copre solo una piccola porzione del cielo dato che non vogliamo che il sole sia troppo grande. Il colore è usato per i valori della mappa compresi tra 0.000 e 0.002 per ottenere un netto contrasto al valore 0.002 (non vogliamo che il sole venga sfumato nel cielo). Il rosso più scuro usato per l'alone si sfuma nel blu scuro dal valore 0.002 a 0.200. Tutti i valori sopra a 0.200 corrispondono al blu scuro. Il comando `rotate -135*x` è usato per ruotare il sole e tutto il cielo alla posizione definitiva. Senza questa rotazione il sole sarebbe a 0°, cioè sotto di noi.
Osservando l'immagine risultante vedrai gli effetti che si possono ottenere con la sfera celeste.

4.10.2.3 Aggiungere Qualche Nuvola

Per migliorare ulteriormente la nostra immagine , vogliamo aggiungere qualche nuvola, con un secondo pigmento. Questo nuovo pigmento usa il motivo `bozo` per creare le nuvolette. Dal momento che si troverà in cima agli altri strati di pigmento avrà bisogno di alcuni colori trasparenti (guardate le componenti da 0.5 a 1.0).

```
sky_sphere {
pigment {
gradient y
color_map {
[0.000 0.002 color rgb <1.0, 0.2, 0.0>
color rgb <1.0, 0.2, 0.0>]
[0.002 0.200 color rgb <0.8, 0.1, 0.0>
color rgb <0.2, 0.2, 0.3>]
}
scale 2
translate -1
```

```
}
pigment {
bozo
turbulence 0.65
octaves 6
omega 0.7
lambda 2
color_map {
[0.0 0.1 color rgb <0.85, 0.85, 0.85>
color rgb <0.75, 0.75, 0.75>]
[0.1 0.5 color rgb <0.75, 0.75, 0.75>
color rgbt <1, 1, 1, 1>]
[0.5 1.0 color rgbt <1, 1, 1, 1>
color rgbt <1, 1, 1, 1>]
}
scale <0.2, 0.5, 0.2>
}
rotate -135*x
}
```

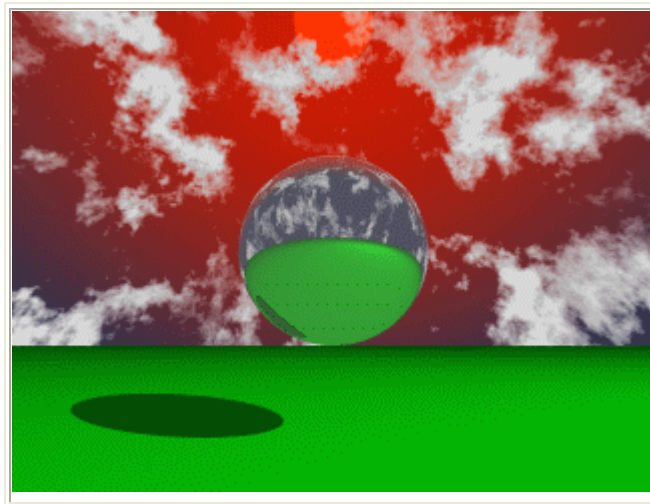



Fig. 177-Aggiungere le nuvole

La sfera celeste ha un difetto, come vi sarete accorti guardando l'immagine finale (**skysph3.pov**). il sole non emette luce e le nuvole non proiettano ombre. Se vuoi che le nuvole proiettino le ombre dovrai usare una grande sfera con una texture appropriata e una luce fuori dalla sfera.

4.10.3 La Nebbia

Puoi usare la funzione `fog` per aggiungere due tipi diversi di nebbia alla tua scena : la nebbia costante e la nebbia raso terra. La nebbia costante ha una densità uguale in ogni punto, mentre la densità della nebbia raso terra diminuirà salendo.
L'uso di questi due tipi di nebbia verrà descritto approfonditamente nei prossimi paragrafi.

4.10.3.1 La Nebbia Costante

Il tipo di nebbia più facile da usare è la nebbia costante, che ha una densità uguale in ogni punto. E' determinata dalla parola chiave `distance` che imposta la densità ed il colore della nebbia.
Il valore assegnato alla distanza determina la distanza alla quale il 36.8% dello sfondo è ancora visibile (per una spiegazione più dettagliata vedere il paragrafo "Nebbia").
Il colore della nebbia può essere usato per creare qualunque colore dal bianco al rosso. Si può anche

usare una nebbia nera per simulare un limitato campo visivo.
L'esempio seguente mostra come aggiungere una semplice nebbia ad una scena (**fog1.pov**).

```
#include "colors.inc"
camera {
location <0, 20, -100>
}

background { colour SkyBlue }

plane { y, -10
pigment {
checker colour Yellow colour Green
scale 20
}
}

sphere { <0, 25, 0>, 40
pigment { Red }
finish { phong 1.0 phong_size 20 }
}

sphere { <-100, 150, 200>, 20
pigment { Green }
finish { phong 1.0 phong_size 20 }
}

sphere { <100, 25, 100>, 30
pigment { Blue }
finish { phong 1.0 phong_size 20 }
}

light_source { <100, 120, 40> colour White}

fog {
distance 150
colour rgb<0.3, 0.5, 0.2>
}
```

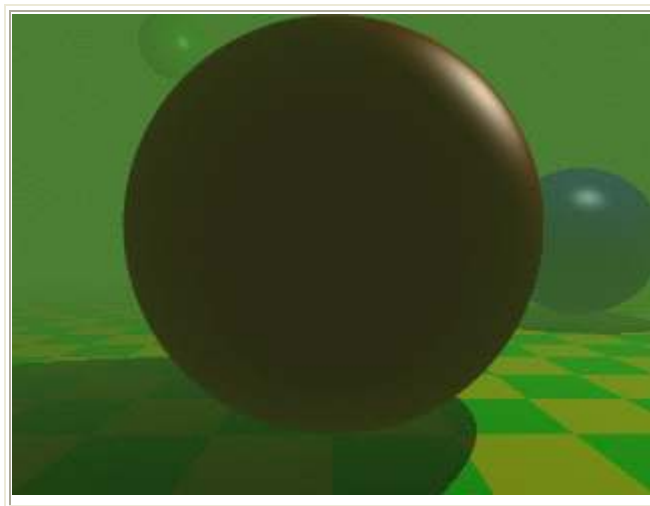



Fig. 178-Nebbia costante

In relazione alla loro distanza le sfere di questa scena scompaiono più o meno nella nebbia verdastra, allo stesso modo si comporta il piano.

4.10.3.2 Impostare una Trasparenza Minima

Se vuoi assicurarti che lo sfondo non scompaia completamente nella nebbia puoi impostare il canale della trasmittanza del colore della nebbia alla quantità di sfondo che vuoi che rimanga sempre visibile. Usando un valore di trasmittanza di 0.2 come in

```
fog {
distance 150
colour rgbt<0.3, 0.5, 0.2, 0.2>
}
```

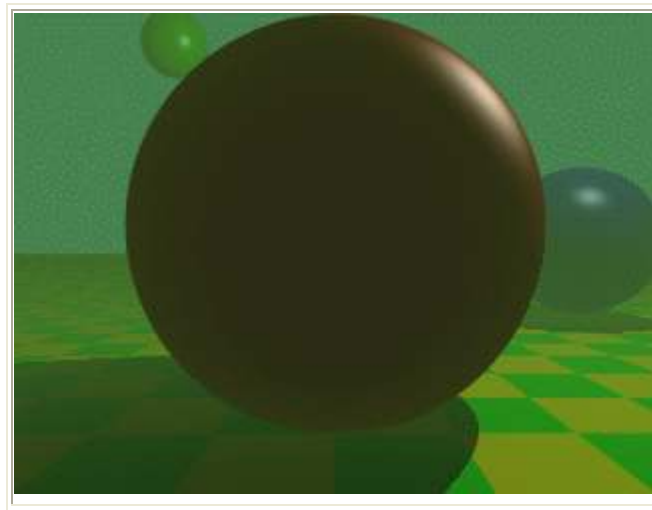



Fig. 179-Trasparenza della nebbia

la trasparenza della nebbia non scende mai al di sotto del 20% come puoi vedere dall'immagine risultante (**fog2.pov**).

4.10.3.3 Creare una Nebbia Filtrante

La nebbia verdastra che abbiamo usato fino ad ora non filtra la luce che le passa attraverso. Tutto ciò che fa è diminuire l'intensità della luce. Possiamo modificare questo comportamento usando un valore diverso da zero per la componente di filtro del colore della nebbia (**fog3.pov**).

```
fog {
distance 150
colour rgbf<0.3, 0.5, 0.2, 1.0>
}
```

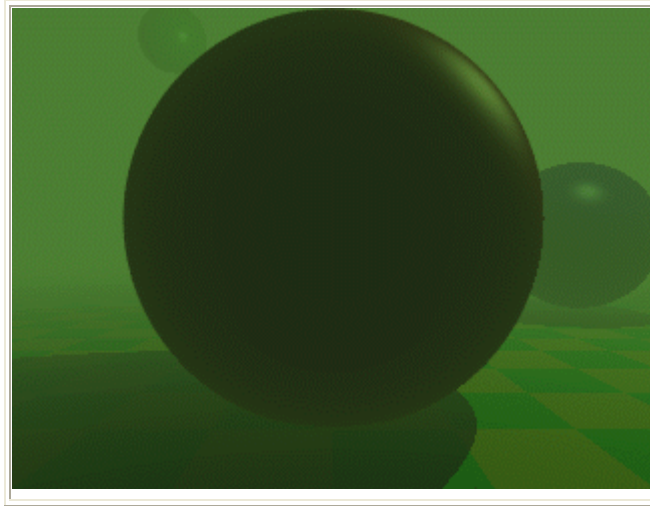



Fig. 180-Nebbia filtrante

Il valore del filtro determina la quantità di luce che viene filtrata dalla nebbia. Nel nostro esempio il 100% della luce che passa attraverso la nebbia sarà filtrata. Se avessimo usato un valore di 0.7 solo il 70% della luce sarebbe stata filtrata il restante 30% avrebbe attraversato la nebbia senza venire filtrato.
Noterai che l'intensità degli oggetti nella nebbia non solo è diminuita a causa del colore della nebbia, ma che i colori stessi sono influenzati dalla nebbia. La sfera rossa e specialmente quella blu hanno assunto una tonalità verde.

4.10.3.4 Aggiungere Turbolenza alla Nebbia

Per rendere la nostra nebbia più interessante possiamo aggiungere un po' di turbolenza rendendola meno omogenea (vedi **fog4.pov**).

```
fog {
distance 150
colour rgbf<0.3, 0.5, 0.2, 1.0>
turbulence 0.2
turb_depth 0.3
}
```

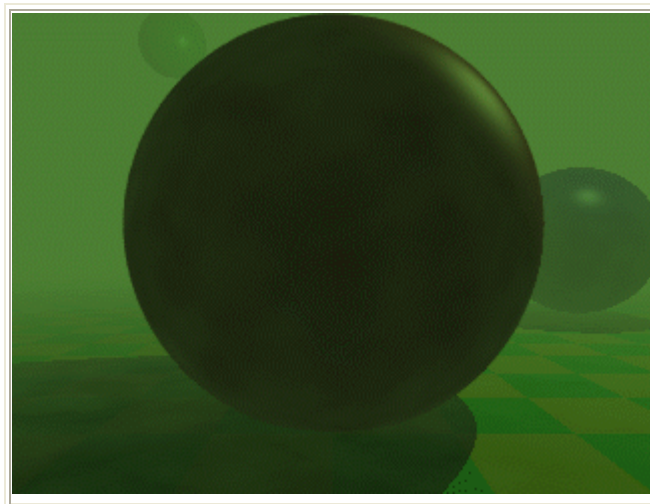



Fig. 181-Aggiungere un pò di turbolenza alla nebbia

La parola chiave `turbulence` serve a specificare la quantità di turbolenza usata mentre il valore di `turb_depth` serve a spostare il punto in cui la turbolenza viene calcolata lungo il raggio visivo. Valori vicini a 0 spostano il punto verso l'osservatore, mentre valori vicini ad 1 lo spostano verso il punto di intersezione. Questo parametro può essere utilizzato per evitare disturbi che possono verificarsi nella nebbia a causa della turbolenza (ciò accade normalmente a punti di intersezione molto lontani, specialmente se non c'è intersezione, cioè se viene colpito lo sfondo). Nel caso che si ottengano immagini disturbate conviene abbassare il valore di `turb_depth` fino a quando il disturbo sparisce. Si dovrebbe ricordare che l'effettiva densità della nebbia non viene modificata. Solo il valore di attenuazione basato sulla distanza viene modificato dalla turbolenza in un punto lungo il raggio visivo.

4.10.3.5 Nebbia Raso Terra

Il tipo di nebbia più interessante e versatile è la nebbia raso terra, che viene selezionata col comando `fog_type`. Il suo aspetto è descritto dalle parole chiave `fog_offset` e `fog_alt`. `Fog_offset` specifica l'altezza, cioè il valore delle y sotto il quale la nebbia ha una densità costante di 1. La parola chiave `fog_alt` determina quanto velocemente la densità della nebbia arriva a 0 via via che ci si sposta lungo l'asse y. All'altezza `fog_offset+fog_alt` la nebbia avrà densità del 25%. Il seguente esempio (**fog5.pov**) usa una nebbia che ha densità costante sotto y=25 (il centro della sfera rossa) e scende rapidamente col salire della quota.

```
fog {
distance 150
colour rgbf<0.3, 0.5, 0.2, 1.0>
fog_type 2
fog_offset 25
fog_alt 1
}
```

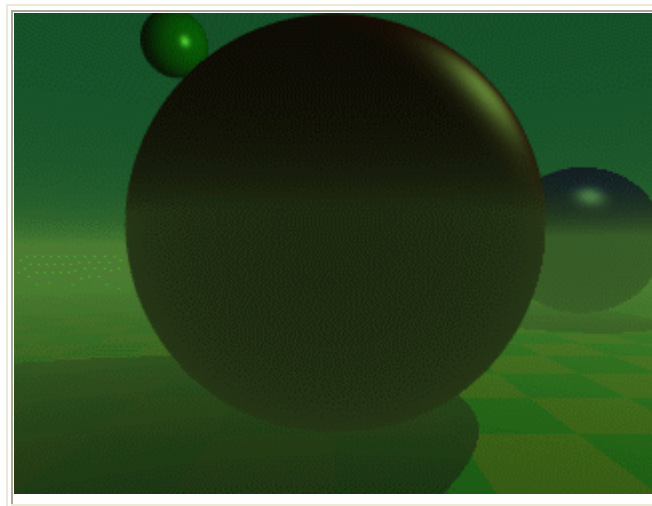



Fig. 182-Nebbia raso terra

4.10.3.6 Strati Multipli di Nebbia

E' possibile usare diversi strati di nebbia usando più di una frase `fog{...}` nel file scena. Questo è molto utile se si vogliono ottenere dei buoni effetti con strati di nebbia ai quali è stata aggiunta turbolenza. Per esempio potresti aggiungere diversi strati di nebbie diversamente colorate per dare un tocco di mistero alla tua scena.
Prova il seguente esempio (**fog6.pov**).

```
fog {
distance 150
colour rgb<0.3, 0.5, 0.2>
fog_type 2
fog_offset 25
fog_alt 1
turbulence 0.1
turb_depth 0.2
}

fog {
distance 150
colour rgb<0.5, 0.1, 0.1>
fog_type 2
fog_offset 15
fog_alt 4
turbulence 0.2
turb_depth 0.2
}

fog {
distance 150
colour rgb<0.1, 0.1, 0.6>
fog_type 2
fog_offset 10
fog_alt 2
}
```

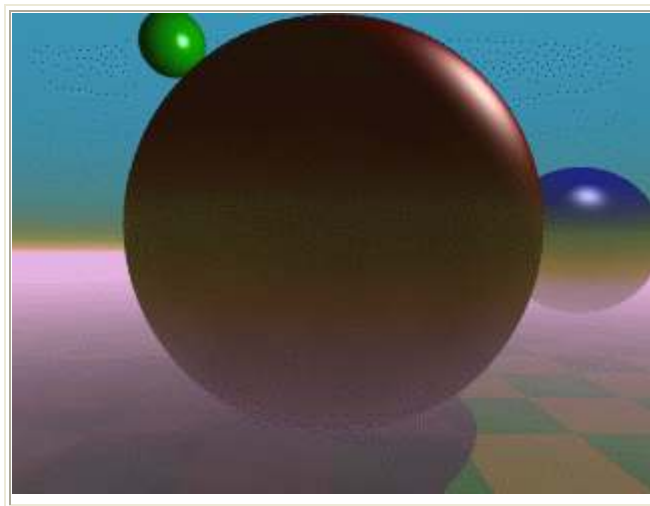



Fig. 183-Più strati di nebbia

Puoi combinare insieme nebbie a densità costante, nebbie raso terra, nebbie filtranti, nebbie non filtranti ecc.

4.10.3.7 Nebbia e Oggetti Vuoti

Tutte le volte che usi la nebbia e la macchina fotografica dentro ad un oggetto non vuoto non otterrai alcun effetto. Per una spiegazione dettagliata del perché ciò accade vedi il paragrafo "Oggetti Vuoti e Oggetti Solidi".
Per evitare questo problema devi rendere tutti questi oggetti vuoti assicurandoti che la macchina

fotografica si trovi al loro esterno (usando la parola chiave `inverse`) o aggiungendogli la parola chiave `hollow` (che è molto più semplice).

4.10.4 L'Atmosfera

Nota Bene : la funzione atmosfera in POV-Ray è in qualche modo ancora allo stato sperimentale. Ci sono buone probabilità che sintassi ed implementazione della funzione cambieranno nelle future versioni. Non possiamo garantire che scene create con POV-Ray 3.0 verranno renderizzate allo stesso modo in versioni successive, né la piena compatibilità all'indietro del linguaggio.

La funzione `atmosphere` può essere usata per modellare l'interazione della luce con le particelle sospese nell'aria. I raggi di luce diventeranno visibili e gli oggetti proietteranno le loro ombre nella nebbia o nella polvere che riempie l'aria. Il modello di atmosfera usato in POV-Ray assume una distribuzione costante delle particelle ovunque, tranne che all'interno degli oggetti. Se vuoi creare nuvole simili a nebbia o fumo, dovrai usare la funzione 'Halo' descritta nel paragrafo "Aloni".

4.10.4.1 Iniziare con una Stanza Vuota

Vogliamo creare una semplice scena per spiegare come funziona la funzione atmosfera e come ottenere i migliori risultati.
Immagina una stanza con una finestra. La luce passa attraverso la finestra e viene dispersa dalle particelle di luce nell'aria. Vedrai quindi raggi di luce che provengono dalla finestra e illuminano il pavimento.
Vogliamo modellare questa scena passo dopo passo. Il seguente esempio inizia dalla stanza, la finestra ed una luce in qualche punto all'esterno. Per adesso, non c'è l'atmosfera che permette di verificare se l'illuminazione è corretta (vedi **atmos1.pov**)

```
camera {
location <-10, 8, -19>
look_at <0, 5, 0>
angle 75
}

background { color rgb <0.2, 0.4, 0.8> }

light_source { <0, 19, 0> color rgb 0.5 atmosphere off }

light_source {
<40, 25, 0> color rgb <1, 1, 1>
spotlight
point_at <0, 5, 0>
radius 20
falloff 20
atmospheric_attenuation on
}

union {
difference {
box { <-21, -1, -21>, <21, 21, 21> }
box { <-20, 0, -20>, <20, 20, 20> }
box { <19.9, 5, -3>, <21.1, 15, 3> }
}
box { <20, 5, -0.25>, <21, 15, 0.25> }
```

```
box { <20, 9.775, -3>, <21, 10.25, 3> }
pigment { color red 1 green 1 blue 1 }
finish { ambient 0.2 diffuse 0.5 }
}
```

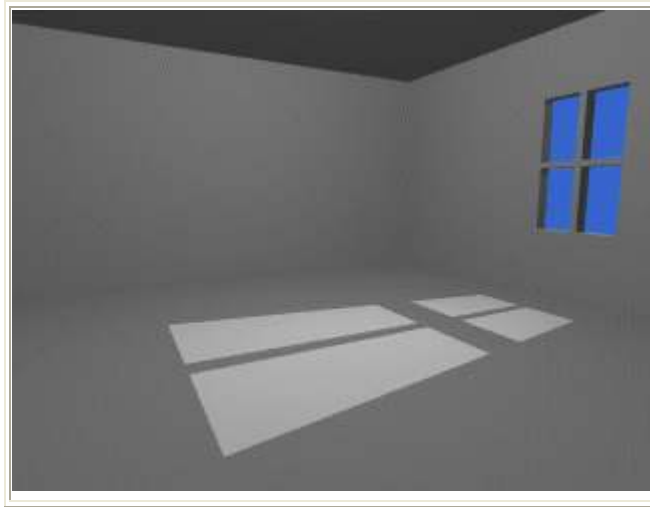



Fig. 183-Stanza

La luce puntiforme è usata per illuminare la stanza dall'interno senza alcuna interazione con l'atmosfera. Questo effeto è ottenuto aggiungendo il comando `atmosphere off`. Non abbiamo bisogno di occuparci di questa luce quando aggiungiamo l'atmosfera. La luce spot è invece caratterizzata dalla parola chiave `atmospheric_attenuation`. Ciò significa che l'intensità della luce che proviene da questo spot sarà diminuita dall'atmosfera.
L'oggetto unione è usato per modellare la stanza e la finestra. Dato che usiamo una differenza tra due parallelepipedi per modellare la stanza (le prime due `box` nella frase `difference{...}`) non c'è bisogno di rendere vuota l'unione con la parola chiave `hollow`. Se siamo all'interno della stanza ci troviamo in effetti al di fuori dell'oggetto (vedi il paragrafo "Oggetti Vuoti ed Atmosfera")

4.10.4.2 Aggiungere Polvere alla Stanza

Il passo successivo è di aggiungere polvere alla stanza. L'effetto si ottiene con le seguenti linee (vedi **atmos2.pov**).

```
atmosphere {
type 1
samples 10
distance 40
scattering 0.2
}
```

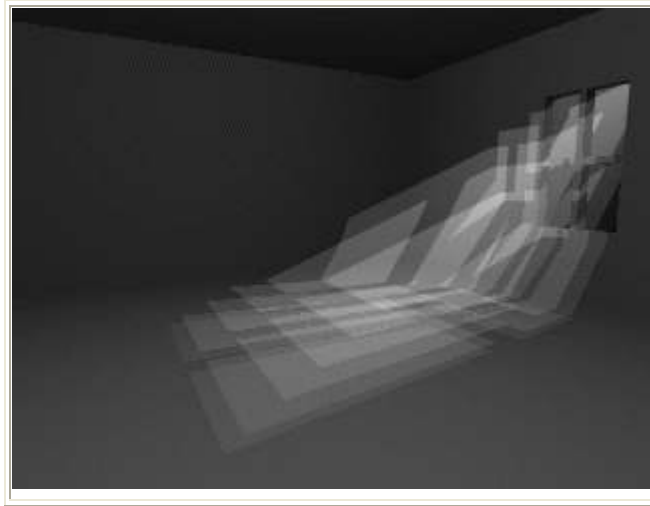



Fig. 184-Polvere

la parola chiave `type` seleziona il tipo di dispersione della luce che vogliamo usare. In questo caso usiamo la dispersione isotropa, che disperde la luce ugualmente in tutte le direzioni (vedi "Atmosfera" per maggiori dettagli).
La parola chiave `samples` determina il numero di campioni usati per accumulare l'effetto atmosferico. Se il risultato non è ciò che ti aspettavi, puoi migliorarlo aumentando il numero di campioni. Il problema di scegliere un buon tasso di campionamento è un compromesso tra la qualità dell'immagine ed il tempo di rendering. Un alto tasso di campionamento funzionerà quasi sempre, ma aumenterà anche il tempo di rendering. Questo è un parametro su cui sperimentare.
La parola chiave `distance` specifica la densità dell'atmosfera. Funziona allo stesso modo del parametro `distance` della funzione nebbia.
Infine, il valore assegnato al parametro `scattering` determina la quantità di luce che viene diffusa dalle particelle (il resto è assorbito). Come vedremo più avanti, questo parametro è molto utile per determinare l'aspetto complessivo dell'atmosfera.
Osservando l'immagine ottenuta dalla scena precedente, noterai alcune artificiosità molto brutte, conosciute come *bande di Mach*. Sono il risultato di un tasso di campionamento molto basso. Il modo di evitarle è descritto nel paragrafo seguente.

4.10.4.3 Scegliere un buon Tasso di Campionamento

Come hai appena visto, un tasso di campionamento troppo basso può causare dei risultati sgradevoli. Ci sono diversi metodi per ridurre od addirittura evitare questi problemi. L'approccio brutale è aumentare il tasso di campionamento fino a quando l'artificiosità svanisce ed ottenere così un'immagine soddisfacente. Per quanto questo funzioni sempre, è una cattiva idea, poiché allunga molto i tempi di rendering. Un approccio migliore è usare prima il jittering e l'anti-aliasing. Se nessuna di queste due funzioni aiuta, dovrai aumentare il tasso di campionamento.
Il jittering sposta ogni punto di una quantità piccola e casuale lungo la direzione del campionamento. Ciò aiuta a ridurre le bande regolari che sono un risultato dell'aliasing, il che aiuta molto dato che l'occhio umano è molto più sensibile alle bande regolari che non a un 'rumore' disperso nell'immagine.
Usa la parola chiave `jitter` seguita dalla quantità che desideri. Valori ottimali sono numeri decimali minori di 0.5, dato che valori più alti risultano in un'immagine troppo disturbata. Ricorda che il jittering non elimina le artificiosità introdotte da un tasso di campionamento troppo basso, ma aiuta solo a renderle meno visibili.
Un altro, migliore metodo di ridurre l'aliasing è di usare il sovracampionamento adattivo. Questo metodo lancia raggi addizionali dove è probabile che ce ne sia bisogno. Se l'intensità del colore tra due campioni adiacenti è troppo diversa (leggi : la loro differenza è al di sopra di un determinato

valore di soglia), vengono tracciati campioni aggiuntivi in mezzo ai due precedenti. Questo metodo è eseguito ricorsivamente fino ad un determinato numero di iterazioni o fino a quando i valori riportati dai campioni non sono più vicini. Le parole chiave `aa_level` ed `aa_threshold` danno il pieno controllo sul processo di sovracampionamento. La parola chiave `aa_level` determina il massimo numero di iterazioni mentre `aa_threshold` specifica la differenza massima che si può verificare tra due campioni adiacenti prima che venga effettuato il sovracampionamento.
Dopo tutta questa teoria, torniamo alla nostra scena ed aggiungiamo le appropriate parole chiave per usare sia il jittering che il sovracampionamento (vedi **atmos3.pov**).

```
atmosphere {
type 1
samples 50
distance 40
scattering 0.2
aa_level 4
aa_threshold 0.1
jitter 0.2
}
```

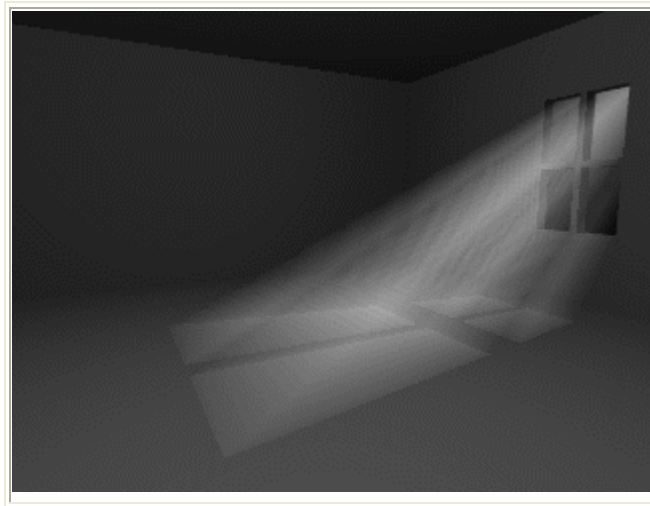



Fig. 185

Se osservi il risultato che ottieni dopo avere aggiunto jittering e sovracampionamento non sarai soddisfatto. L'unico modo di ridurre le artificiosità ancora visibili è aumentare il tasso di sovracampionamento scegliendo un numero più alto di campioni (aumentando quindi la voce `samples` nella definizione dell'atmosfera).
In questo modo otterrai un buon risultato, rendendo (quasi) invisibili le artificiosità. A proposito, la quantità di polvere nella nostra stanza forse è un po' esagerata, ma questo è solo un esempio. E gli esempi, a volte, tendono ad essere esagerati.

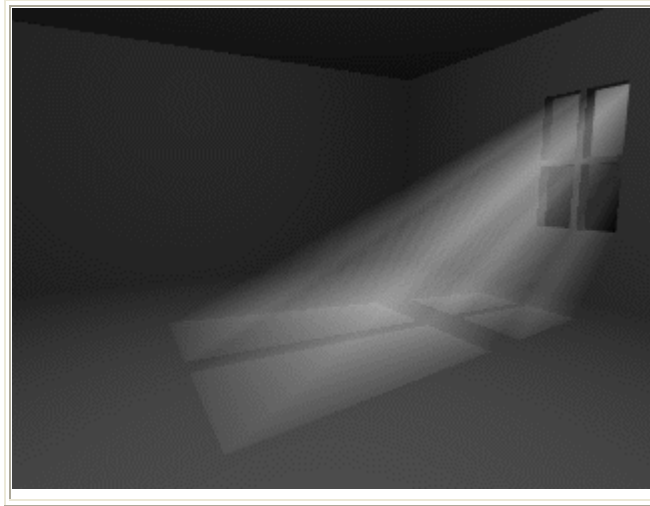



Fig. 186-Aumentare il numero dei campioni

4.10.4.4 Usare Atmosfere Colorate

Puoi assegnare un colore all'atmosfera, in modo da avere un maggiore controllo sul suo aspetto. Per prima cosa, il colore filtra tutta la luce che attraversa l'atmosfera, proveniente da sorgenti luminose, raggi riflessi o rifratti, o dallo sfondo. La quantità di luce che viene filtrata è determinata dal valore di filtro impostato nel colore. Un valore di zero per il filtro significa che la luce non viene influenzata dal colore, mentre un valore di 1 significa che tutta la luce è filtrata dall'atmosfera.
Se vuoi creare un atmosfera rossastra, per esempio, puoi aggiungere la seguente linea nella frase `atmosphere{...}` che abbiamo usato nell'esempio sopra.

```
color rgbf <1, 0, 0, 0.25>
```

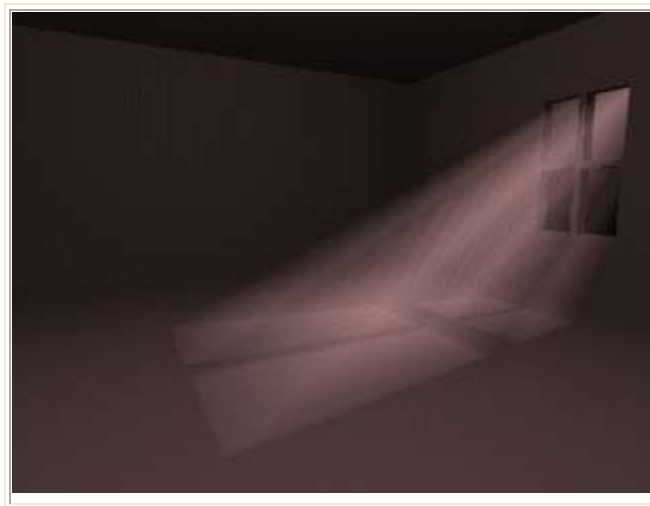



Fig. 187-Atmosfera colorata

Usare solo `rgb <1, 0, 0>` non funziona perché il valore del filtro sarebbe zero e quindi la luce non verrebbe filtrata dal colore. Un valore di filtro 0.25 significa che il 25% della luce che passa attraverso l'atmosfera verrà filtrata dal colore rosso ed il 75% passerà inalterato.
Il canale di trasmittanza dell'atmosfera serve invece per specificare una trasparenza minima. Per default il canale della trasmittanza è a zero e quindi non c'è trasparenza. Usare un valore positivo nel canale di trasmittanza del colore dell'atmosfera ti permette di determinare la quantità di luce proveniente dallo sfondo che attraverserà comunque l'atmosfera, indipendentemente dal suo spessore, definito con la parola chiave `thickness`.

Se usi, per esempio, un colore `rgbt <0, 0, 0, 3>` nel nostro esempio, puoi rendere visibile lo sfondo blu (che finora era nascosto dall'atmosfera).

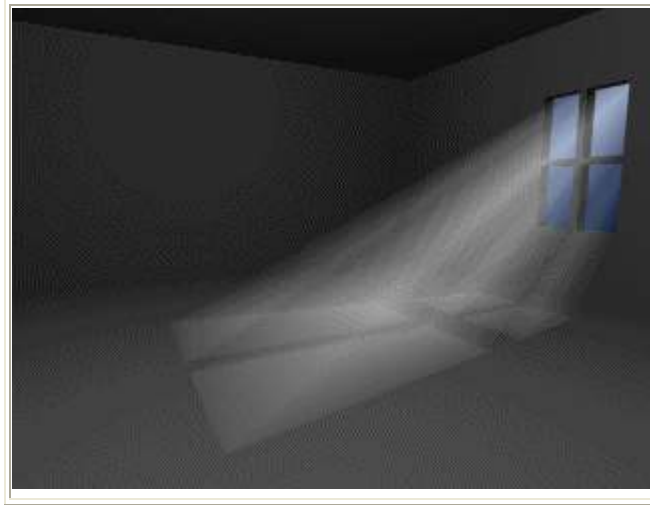



Fig. 188

4.10.4.5 Trucchi sull'Atmosfera

E' molto difficile ottenere buoni risultati usando la funzione atmosfera. Alcuni dei problemi più comuni saranno discussi nei prossimi paragrafi per aiutarti a risolverli. (vedi anche l'appendice G.5 - "Domande sull'Atmosfera").

4.10.4.5.1 Scegliere la Distanza e i Parametri di Dispersione

Il primo passo difficile nel creare una buona atmosfera è scegliere i valori di distanza e dispersione. Ciò permette di controllare la visibilità degli oggetti nella scena e gli effetti atmosferici.
L'approccio migliore è scegliere prima il valore della distanza. Questo valore determina la visibilità degli oggetti nella scena, indipendentemente dalla dispersione della luce da parte dell'atmosfera. Funziona nello stesso modo del parametro `distance` usato per la nebbia.
Dato che la nebbia è molto simile all'atmosfera non illuminata, puoi usare una nebbia al posto dell'atmosfera per scegliere un valore ottimale per `distance`. Se dovessi farlo per la stanza che abbiamo usato prima, utilizzeresti la seguente frase `fog{...}` al posto dell'atmosfera (vedi **atmos4.pov**)

```
fog {
distance 40
color rgb <0, 0, 0>
}
```

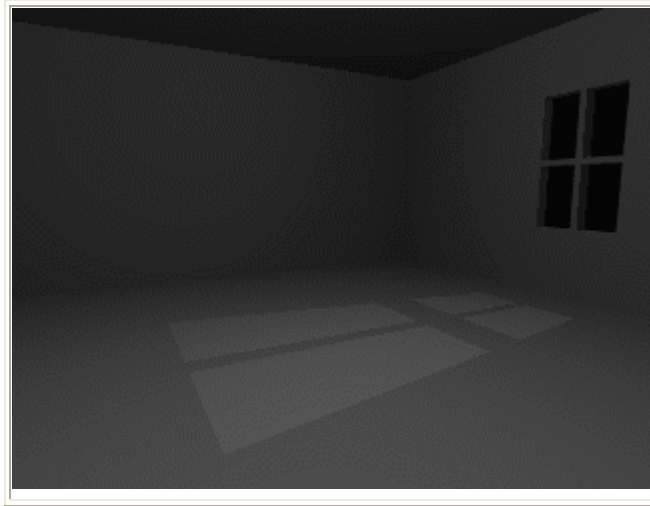



Fig. 189

Il colore nero è usato per simulare l'attenuazione della luce che otterrai in quelle parti della scena che si trovano in ombra.
Se vuoi usare un'atmosfera colorata dovrai usare per la nebbia lo stesso colore che vuoi usare per l'atmosfera, compresi i canali di filtro e trasmittanza (vedi "Usare Atmosfere Colorate" e "Atmosfera" per una spiegazione sul colore dell'atmosfera).
Se vuoi simulare (grossolanamente) l'aspetto di quelle parti della scena illuminate da una sorgente luminosa, puoi invece usare il colore dell'atmosfera all'interno della nebbia.
Quando sei soddisfatto del valore di `distance` dovrai scegliere un valore per la dispersione (parola chiave `scattering`). Questo valore ti permette di adattare la densità dell'atmosfera alle tue necessità. Iniziando con un valore di 1, dovrai aumentarlo se gli effetti dati dall'atmosfera sono appena visibili. Viceversa dovrai diminuirlo se non vedi nulla nelle zone illuminate dell'atmosfera. Devi fare attenzione al fatto che potresti dovere impostare valori molto grandi o molto piccoli per ottenere il risultato corretto.

4.10.4.5.2 Atmosfera e Sorgenti Luminose

I risultati migliori si ottengono con luci spot e cilindriche. Queste creano fasci di luce di grande effetto e sono veloci da renderizzare perché il campionamento dell'atmosfera avviene solo all'interno del cono di luce dello spot (o del cilindro di luce della luce cilindrica !).
Se vuoi aggiungere una sorgente luminosa che non interagisca con l'atmosfera puoi usare la parola chiave `atmosphere` dentro la frase di descrizione della sorgente luminosa (vedi "Interazione con l'Atmosfera"). Basta aggiungere `atmosphere off`.
Come impostazione predefinita, la luce che proviene da qualunque sorgente luminosa non verrà attenuata dall'atmosfera. In questo modo, i punti illuminati direttamente saranno di solito troppo luminosi. Questo effetto può essere evitato con `atmospheric_attenuation on`.

4.10.4.5.3 Tipi di Dispersione Atmosferica

I diversi modelli di dispersione elencati nel paragrafo "Atmosfera" possono essere usati per simulare differenti tipi di particelle. Questa è un'opzione che richiede qualche esperimento.
La dispersione di Rayleigh è usata per piccole particelle come polvere e fumo, mentre la dispersione di Mie è la migliore per simulare la nebbia. Se hai visto la scena del faro nel film ***Casper*** hai già visto che effetto ha questo tipo di dispersione : in questa scena il raggio di luce del faro diventa visibile quando è diretto verso l'osservatore. Quando si rivolge altrove, svanisce. Questo comportamento della luce è tipico per minuscole gocce d'acqua, come quelle che vengono simulate dalla dispersione di Mie.

4.10.4.5.4 Aumentare la Risoluzione dell'Immagine

Devi fare attenzione al fatto che potresti essere obbligato ad aumentare il tasso di campionamento dell'atmosfera se vuoi aumentare la risoluzione dell'immagine. Altrimenti, alcune artificiosità dovute a fenomeni di aliasing che non erano visibili a risoluzioni basse possono comparire all'improvviso.

4.10.4.5.5 Oggetti Vuoti ed Atmosfera

Quando usi la funzione atmosfera, devi fare attenzione che tutti gli oggetti che dovrebbero contenere l'atmosfera siano resi cavi usando la parola chiave `hollow`. Anche se non è ovvio, ciò vale anche per oggetti infiniti e superfici come quadriche, quartiche, triangoli, poligoni ecc. Quando utilizzi uno di questi oggetti dovresti sempre aggiungergli la parola chiave `hollow` a meno che tu non sia assolutamente sicuro di non averne bisogno. Devi anche assicurarti che siano cavi tutti gli oggetti all'interno dei quali si trova la macchina fotografica. Tutte le volte che ottieni strani risultati controlla gli oggetti ed eventualmente aggiungi loro la parola chiave `hollow`.

4.10.5 L'Arcobaleno

La funzione arcobaleno può essere usata per creare arcobaleni ed effetti più strani. L'arcobaleno è un effetto simile a quello della nebbia, ma che viene ristretto ad un volume conico.

4.10.5.1 Iniziare con un Arcobaleno Semplice

L'arcobaleno è specificato per mezzo di molti parametri : l'angolo al quale è visibile, l'ampiezza della banda colorata, la direzione della luce proveniente, la distanza (simile a quella specificata per la nebbia) ed infine la mappatura dei colori da usare.
Dimensione e forma dell'arcobaleno sono determinati dalle parole chiave `angle` e `width`. La parola chiave `direction` serve ad impostare la direzione da cui proviene la luce in modo da determinare la posizione dell'arcobaleno. L'arcobaleno è visibile quando l'angolo tra il vettore direzione e la direzione di provenienza della luce incidente è compreso tra `angle-width/2` e `angle+width/2`.
La luce di cui stiamo parlando è virtuale, non c'è bisogno che ci sia una reale sorgente luminosa per creare l'arcobaleno.
L'arcobaleno è un effetto simile alla nebbia, cioè il colore dell'arcobaleno è mescolato col colore della sfondo in dipendenza dalla distanza col punto di intersezione. Se scegli piccoli valori di distanza l'arcobaleno sarà visibile sugli oggetti, non solo contro lo sfondo. Questo effetto può essere evitato scegliendo grandi valori per `distance`.
La mappa dei colori è la parte critica dell'arcobaleno poiché contiene tutti i colori che possono essere normalmente visti nell'arcobaleno. Il colore della banda più interna è alla posizione 0 mentre quello della banda più esterna è alla posizione 1. Si dovrebbe notare che a causa della limitata varietà di colori che un monitor può normalmente mostrare, è impossibile creare un vero arcobaleno. Ci sono alcuni colori che semplicemente non è possibile mostrare.
Il canale del filtro della mappatura dei colori dell'arcobaleno funziona in modo analogo al canale del filtro che usiamo nella nebbia. Determina quanta parte della luce che attraversa l'arcobaleno è filtrata dal colore.
Il seguente esempio mostra una semplice scena con un piano che rappresenta il suolo ed un arcobaleno (un po') esagerato (**rainbow1.pov**)

```
#include "colors.inc"

camera {
location <0, 20, -100>
look_at <0, 25, 0>
```

```
angle 80
}

background { color SkyBlue }

plane { y, -10 pigment { colour Green } }

light_source {<100, 120, 40> colour White}

// dichiara i colori dell'arcobaleno

#declare r_violet1 = colour rgbf<1.0, 0.5, 1.0, 1.0>
#declare r_violet2 = colour rgbf<1.0, 0.5, 1.0, 0.8>
#declare r_indigo = colour rgbf<0.5, 0.5, 1.0, 0.8>
#declare r_blue = colour rgbf<0.2, 0.2, 1.0, 0.8>
#declare r_cyan = colour rgbf<0.2, 1.0, 1.0, 0.8>
#declare r_green = colour rgbf<0.2, 1.0, 0.2, 0.8>
#declare r_yellow = colour rgbf<1.0, 1.0, 0.2, 0.8>
#declare r_orange = colour rgbf<1.0, 0.5, 0.2, 0.8>
#declare r_red1 = colour rgbf<1.0, 0.2, 0.2, 0.8>
#declare r_red2 = colour rgbf<1.0, 0.2, 0.2, 1.0>

// crea l'arcobaleno

rainbow {
angle 42.5
width 5
distance 1.0e7
direction <-0.2, -0.2, 1>
jitter 0.01
colour_map {
[0.000 colour r_violet1]
[0.100 colour r_violet2]
[0.214 colour r_indigo]
[0.328 colour r_blue]
[0.442 colour r_cyan]
[0.556 colour r_green]
[0.670 colour r_yellow]
[0.784 colour r_orange]
[0.900 colour r_red1]
}
}
```

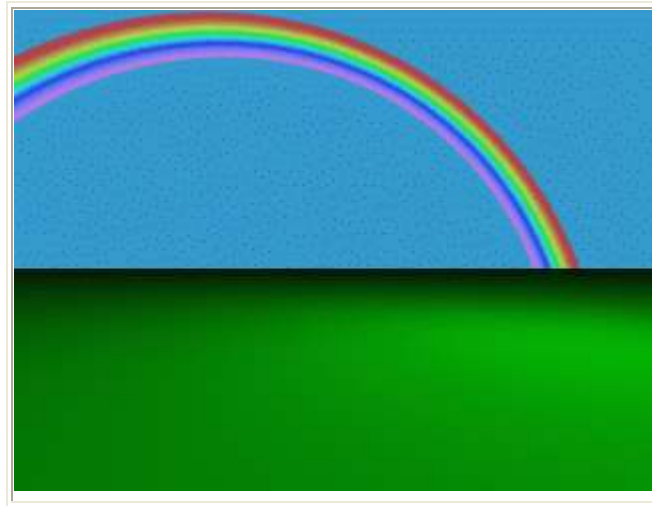



Fig. 190-Arcobaleno

Abbiamo aggiunto qualche irregolarità al colore usando la parola chiave `jitter`.
L'arcobaleno nel nostro esempio è troppo intenso, arcobaleni di questo genere non esistono nella realtà. Possiamo diminuire l'intensità dei colori diminuendo i valori RGB nella mappa.

4.10.5.2 Aumentare la Trasparenza dell'Arcobaleno

Il risultato che abbiamo ottenuto prima è colorato troppo intensamente. Ridurre l'intensità dei colori è di aiuto, ma è molto meglio aumentare la trasparenza dell'arcobaleno, poiché l'effetto riesce più realistico se lo sfondo è visibile attraverso l'arcobaleno.
Possiamo usare il canale della trasmittanza nella mappa dei colori per specificare una trasparenza minima, come abbiamo fatto per la nebbia. Per ottenere risultati realistici, dobbiamo usare valori della trasmittanza molto alti, come si può vedere nel seguente esempio (**rainbow2.pov**)

```
rainbow {
angle 42.5
width 5
distance 1.0e7
direction <-0.2, -0.2, 1>
jitter 0.01
colour_map {
[0.000 colour r_violet1 transmit 0.98]
[0.100 colour r_violet2 transmit 0.96]
[0.214 colour r_indigo transmit 0.94]
[0.328 colour r_blue transmit 0.92]
[0.442 colour r_cyan transmit 0.90]
[0.556 colour r_green transmit 0.92]
[0.670 colour r_yellow transmit 0.94]
[0.784 colour r_orange transmit 0.96]
[0.900 colour r_red1 transmit 0.98]
}
}
```

Il valore della trasmittanza aumenta nelle fasce esterne in modo da sfumare nello sfondo.
L'immagine che ne risulta è molto più realistica della precedente.

Fig. 191-Arcobaleno

4.10.5.3 Usare un Arco di Arcobaleno

Il nostro arcobaleno ha forma ad anello, per quanto la maggior parte di esso si trovi nascosta dietro il piano. Possiamo creare facilmente un arco di arcobaleno mediante la parola chiave `arc_angle`, fornendole come valore un angolo minore di 360°.
Se usiamo ad esempio `arc_angle 120`, otterremo un arco di arcobaleno di 120 gradi che termina bruscamente alle sue estremità. Per evitare ciò, possiamo usare la parola chiave `falloff_angle` per specificare una regione in cui l'arcobaleno si fonda gradualmente con lo sfondo.
Come è spiegato nel paragrafo "Arcobaleno" (vedi § 7.7.5) l'arco si estende da `-arc_angle/2` a `arc_angle/2` mentre la sfumatura si verifica da `-arc_angle/2 a falloff_angle/2` e da `falloff_angle/2` fino a `arc_angle/2`. Questo è il motivo per cui il valore di `falloff_angle` deve essere minore o uguale a quello di `arc_angle`.
Nell'esempio seguente usiamo un arco di 120° con un angolo di falloff di 45° su entrambi i lati dell'arcobaleno (**rainbow3.pov**)

```
rainbow {
angle 42.5
width 5
arc_angle 120
falloff_angle 30
distance 1.0e7
direction <-0.2, -0.2, 1>
jitter 0.01
colour_map {
[0.000 colour r_violet1 transmit 0.98]
[0.100 colour r_violet2 transmit 0.96]
[0.214 colour r_indigo transmit 0.94]
[0.328 colour r_blue transmit 0.92]
[0.442 colour r_cyan transmit 0.90]
[0.556 colour r_green transmit 0.92]
[0.670 colour r_yellow transmit 0.94]
[0.784 colour r_orange transmit 0.96]
[0.900 colour r_red1 transmit 0.98]
}
```

```
}
```

Gli angoli dell'arcobaleno sono misurati sulla direzione verticale dell'arcobaleno che può essere specificata usando la parola chiave `up` seguita da un vettore. Per default, questa direzione è l'asse delle y.

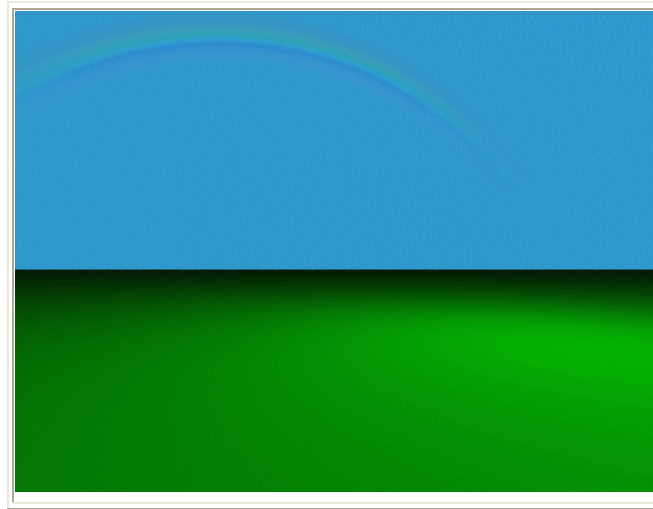



Fig.192-Solo un arco di arcobaleno

4.10.6 Animazione

C'è una moltitudine di programmi disponibili che accettano come input una serie di immagini TGA (come l'output di POV-Ray) e le montano insieme in un'animazione. Questi programmi possono produrre filmati in formato AVI, MPEG, FLI/FLC, od anche GIF animate per uso sul WWW . Il problema quindi non è produrre i filmati, ma i singoli fotogrammi. E qui, ovviamente, interviene POV-Ray. Nelle prime versioni di POV-Ray, produrre un'animazione non era semplice, dato che tutto doveva essere eseguito manualmente. Dovevamo impostare la variabile `clock` e produrre file diversi per ogni fotogramma. Potevamo ottenere una qualche automazione usando file batch o simili espedienti, ma comunque andava fatto tutto a mano ed era un gran lavoro (senza tirare in ballo la frustrazione...immagina cosa succede quando ci si dimentica di impostare nomi di file diversi e, tornando ventiquattr'ore dopo, si scopre che ogni fotogramma aveva soprascritto quello precedente). Ora, finalmente, con POV-Ray 3 abbiamo un modo migliore. Non abbiamo più bisogno di un file batch o di programmi esterni, perché poche semplici impostazioni all'interno di un file .INI appropriato (o nella linea di comando) attiveranno una sequenza interna di animazione (*loop*) che farà si che POV-Ray gestisca automaticamente per noi l'animazione.
In effetti, la discussione del supporto che POV-Ray dà all'animazione può essere divisa in due parti: le impostazioni che dobbiamo regolare nel file .INI (o nella linea di comando) e quelle modifiche che dobbiamo apportare nei nostri file scena. Se abbiamo già lavorato con l'animazione nelle precedenti versioni di POV-Ray, possiamo probabilmente saltare al paragrafo "Impostazioni dei File .ini". Altrimenti, iniziamo con i concetti elementari sull'animazione. Prima di arrivare a come attivare l'anello interno di animazione (*internal animation loop*), vediamo un paio di esempi su come un paio di parole chiave possono far sì che la nostra scena descriva il muoversi degli oggetti col tempo.

4.10.6.1 La Variabile Clock : la Chiave di Tutto

POV-Ray supporta una variabile decimale predefinita chiamata `clock` (tutto a lettere minuscole). Questa è la chiave per automatizzare le animazioni. Nelle operazioni a linea di comando, la variabile `clock` viene impostata fornendo al programma il parametro `+k`. Per esempio, il parametro `+k3.4` imposterebbe il valore di `clock` a 3.4. Allo stesso modo, in un file .INI, la

presenza della linea

```
Clock=3.4
```

darebbe lo stesso risultato.
Se non impostiamo il valore di `clock` e non usiamo il loop di animazione interno a POV-Ray (che verrà descritto tra poco) la variabile `clock` è sempre definita come uguale a zero, quindi è possibile impostare un file scena pensato per l'animazione e renderizzarlo come immagine ferma, durante la fase di progettazione della scena.
L'esempio più semplice per usare questa variabile a nostro vantaggio è definire un oggetto che si sposta a velocità costante, per esempio lungo l'asse delle x. Avremmo quindi la frase

```
translate <clock, 0, 0>
```

nella frase di dichiarazione del nostro oggetto e il loop di animazione assegnerebbe valori progressivamente crescenti a `clock`. Fino a quando vogliamo che solo un oggetto o un aspetto della scena cambi, va tutto bene, ma cosa succede se vogliamo controllare cambiamenti multipli nella nostra scena ? Il segreto è di usare valori *normalizzati* di `clock` (cioè che variano tra zero ed uno) e poi dichiarare altre variabili proporzionali a `clock`. Quindi, impostiamo `clock` (ci stiamo arrivando !) per variare tra 0 ed 1 e poi lo usiamo come fattore moltiplicativo per altre variabili. In questo modo, le altre variabili possono assumere qualunque altro valore e `clock` si mantiene tra 0 ed 1 per qualunque altro utilizzo. Vediamo un esempio (relativamente) semplice.

```
#include "colors.inc"
camera {
location <0, 3, -6>
look_at <0, 0, 0>
}

light_source { <20, 20, -20> color White }

plane { y, 0
pigment { checker color White color Black }
}

sphere { <0, 0, 0> , 1
pigment {
gradient x
color_map {
[0.0 Blue ]
[0.5 Blue ]
[0.5 White ]
[1.0 White ]
}
scale .25
}
rotate <0, 0, -clock*360>
translate <-pi, 1, 0>
translate <2*pi*clock, 0, 0>
}
```

Renderizzando una serie di fotogrammi con la variabile `clock` che varia progressivamente tra zero ed uno, la scena precedente produrrà una palla a strisce che rotola attraverso lo schermo.

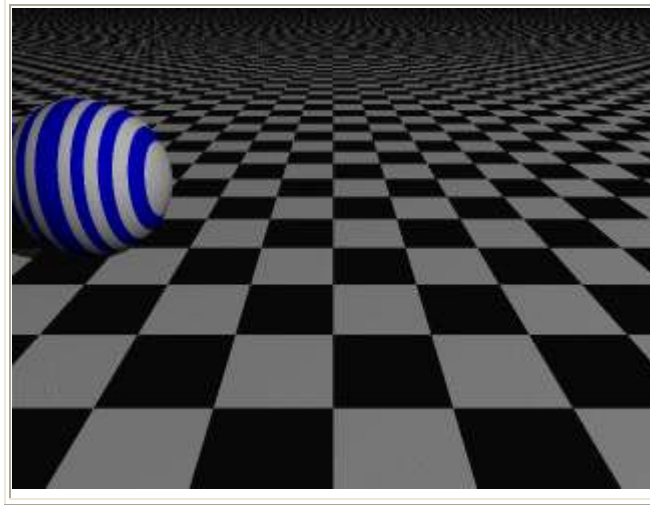



Fig. 193-Semplice animazione

In questo caso abbiamo due obiettivi:

a. Traslare la palla dal punto A al punto B e

b. Ruotare la palla in proporzione al suo movimento lineare, in modo di dare l'impressione che essa stia rotolando e non strisciando.

Partendo dal punto b, iniziamo con la nostra sfera posizionata nell'origine, poiché una rotazione in qualunque altro punto, farebbe orbitare la palla attorno all'origine e non su sé stessa. Attraverso tutta la rotazione, la palla farà un giro completo di 360°, quindi abbiamo usato la formula `360*clock` per determinare la rotazione che si ha in ogni fotogramma. Poiché l'animazione inizia a `clock=0` e finisce a `clock=1`, la rotazione della sfera va da 0 a 360 gradi.
Poi, abbiamo usato la prima traslazione per portare la sfera al punto iniziale (A). Non potremmo averla semplicemente definita in questo punto, dato che avrebbe poi ruotato attorno all'origine, quindi prima di poter completare il punto a (la traslazione) dobbiamo mettere la sfera dove deve trovarsi alla partenza. Poi, la trasliamo nuovamente, questa volta di una distanza dipendente da `clock`, in modo che si sposti dal punto che le abbiamo assegnato per la partenza. Abbiamo scelto la formula `2*pi*r*clock` (la circonferenza massima della sfera moltiplicata per `clock`) in modo che la sfera si sposti di una distanza pari alla sua circonferenza quando ruota di 360 gradi. In questo modo, abbiamo sincronizzato la rotazione della sfera alla sua traslazione, facendo sembrare che essa rotoli tranquillamente sul piano.
Oltre a permetterci di coordinare molteplici aspetti dei cambiamenti che avvengono nella scena durante l'animazione, matematicamente parlando, l'altra buona ragione per utilizzare valori normalizzati di `clock` è che non importa se stiamo renderizzando la nostra animazione per fare una GIF animata da 10 fotogrammi o un filmato AVI da 300. I valori di `clock` sono proporzionali al numero dei fotogrammi e quindi lo stesso codice di POV-Ray funzionerà indipendentemente da quanto è lunga la sequenza. La nostra palla si sposterà esattamente della stessa distanza e non importa quanto finirà con l'essere lunga l'animazione.

4.10.6.2 Variabili Dipendenti da Clock ed Animazioni a più Fasi

Cosa facciamo se vogliamo che la nostra palla rotoli da sinistra a destra per la prima metà dell'animazione, poi cambi direzione di 135 gradi e rotoli da destra a sinistra e indietro per la

seconda metà dell'animazione ? Avremmo bisogno di utilizzare delle istruzioni condizionali che esaminino il valore di `clock` per determinare se è giunto a metà animazione ed in caso positivo comincino a renderizzare una nuova sequenza di animazione, sempre dipendente da `clock`. Ma il nostro scopo, come sopra, è di lavorare con una variabile il cui valore vada da 0 ad 1 (normalizzata) poiché questo rende matematicamente molto più semplice gestire l'animazione di eventi multipli. Quindi, mantenendo le stesse impostazioni per macchina fotografica, luci e piano, facciamo variare `clock` da 0 a 2.
Ora, sostituiamo la sfera dell'esempio precedente con...

```
#if ( clock <= 1 )
sphere { <0, 0, 0> , 1
pigment {
gradient x
color_map {
[0.0 Blue ]
[0.5 Blue ]
[0.5 White ]
[1.0 White ]
}
scale .25
}
rotate <0, 0, -clock*360>
translate <-pi, 1, 0>
translate <2*pi*clock, 0, 0>
}
#else
// (se clock è > 1, siamo nella seconda fase)
// ma vogliamo che vari ancora tra 0 ed 1

#declare ElseClock = clock - 1

sphere { <0, 0, 0> , 1
pigment {
gradient x
color_map {
[0.0 Blue ]
[0.5 Blue ]
[0.5 White ]
[1.0 White ]
}
scale .25
}
rotate <0, 0, ElseClock*360>
translate <-2*pi*ElseClock, 0, 0>
rotate <0, 45, 0>
translate <pi, 1, 0>
}
#else
#end
```

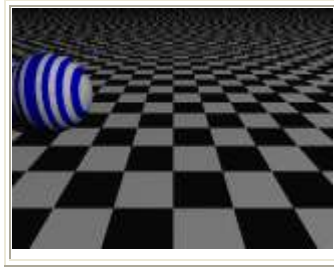



Fig. 194-Palla schizofrenica

Se hai notato che in questo modo la palla farà un'irreale curva ad angolo, quando cambia direzione, complimenti ! Sei indubbiamente un animatore nato. Comunque, per semplificare l'esempio ignoreremo questo aspetto. Sarà abbastanza semplice aggiustarlo una volta esaminato il funzionamento delle istruzioni precedenti.

Tutto ciò che abbiamo fatto è stato assumere che `clock` era compreso tra 0 e 2, mentre vogliamo lavorare con un valore normalizzato. Quindi, quando `clock` supera 1, POV-Ray assume che è iniziata la seconda parte dell'animazione e noi dichiariamo un nuova variabile, detta `Elseclock`, che rendiamo dipendente da `clock`, in modo che mentre `clock` varia tra 1 e 2, `Elseclock` varia tra 0 ed 1. Quindi, anche se c'è un solo `clock`, possiamo avere quante variabili vogliamo (e possiamo permetterci in termini di memoria), così anche in scene molto complesse, la sola variabile `clock` può essere il direttore d'orchestra di tutti gli altri movimenti.

4.10.6.3 La Fase

C'è un'altra parola chiave che dovremmo conoscere per l'animazione : la parola chiave `phase` (fase). Questa parola chiave può essere usata su molti elementi appartenenti a texture, soprattutto quelli che possono utilizzare una mappatura di colore, pigmento, normali o texture. Ricordiamo la sintassi che hanno queste mappe. Ad esempio,

```
color_map {
[0.00 White ]
[0.25 Blue ]
[0.76 Green ]
[1.00 Red ]
}
```

Il valore decimale a sinistra aiuta POV-Ray a mappare il colore sulle varie superfici dell'oggetto. Hai notato che i valori vanno da 0.0 ad 1.0 ?
La parola chiave `phase` fa sì che il colore venga spostato lungo la mappa di un valore decimale che segue la parola chiave. Ora, se stiamo utilizzando un `clock` normalizzato, possiamo fare in modo che quel valore decimale sia proprio la variabile `clock` e il motivo varierà uniformemente col procedere dell'animazione. Vediamo un semplice esempio che utilizza un motivo a gradiente di normali.

```
#include "colors.inc"
#include "textures.inc"

#background { rgb<0.8, 0.8, 0.8> }

camera {
location <1.5, 1, -30>
```

```
look_at <0, 1, 0>
angle 10
}

light_source { <-100, 20, -100> color White }

// bandiera

polygon { 5, <0, 0>, <0, 1>, <1, 1>, <1, 0>, <0, 0>
pigment { Blue }
normal {
gradient x
phase clock
scale <0.2, 1, 1>
sine_wave
}
scale <3, 2, 1>
translate <-1.5, 0, 0>
}

// asta

cylinder { <-1.5, -4, 0>, <-1.5, 2.25, 0>, 0.05
texture { Silver_Metal }
}

// pomo dell'asta

sphere { <-1.5, 2.25, 0>, 0.1
texture { Silver_Metal }
}
```

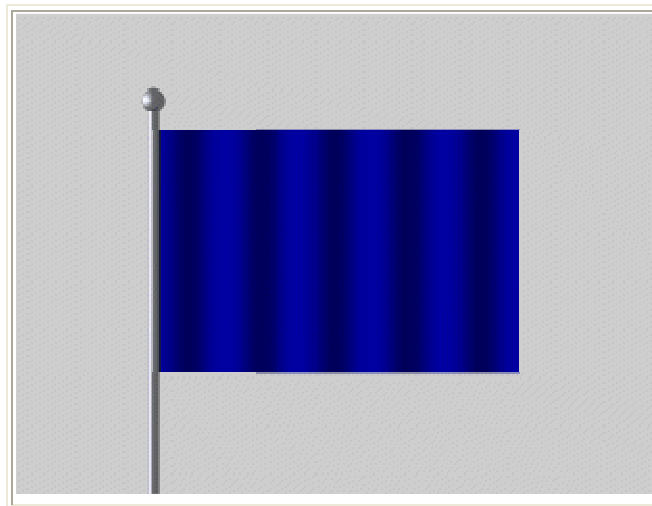



Fig.195-Semplice animazione di una bandiera

Ora abbiamo creato una semplice bandiera blu con un gradiente di normali. Abbiamo fatto usare al gradiente un'onda sinusoidale in modo che la bandiera appaia sventolare. Ma la vera magia qui è la parola chiave `phase`. E' stata impostata per assumere il valore di `clock` come un valore decimale che, via via che `clock` incrementa da 0 ad 1, farà ondeggiare le increspature della bandiera lungo

l'asse delle x. Effettivamente, quando animiamo la bandiera, sembra proprio che essa sventoli. Questo è solo un semplice esempio di come un cambiamento di fase dipendente da clock possa creare interessanti effetti di animazione. Provando `phase` su tutte le texture che abbiamo visto finora, è sorprendente la varietà di effetti di animazione che possiamo creare, senza effettivamente dover nemmeno muovere l'oggetto.

4.10.6.4 Non Usare Jitter o Crand

Un'ultima informazione per risparmiarti frustrazione. Dato che `jitter` è uno degli elementi dell'anti-aliasing, potremmo limitarci a citarlo nel capitolo sulle informazioni nei file **.ini**, ma l'anti-aliasing interviene anche nelle area light e nei nuovi effetti atmosferici di POV-Ray, quindi questa è un occasione buona come altre.
Il jittering è una piccolissima perturbazione che viene introdotta nei raggi luminosi, per diffondere e disperdere i piccoli errori di aliasing che non potrebbero essere eliminati in altro modo, anche con alti livelli di anti-aliasing. Disponendo in maniera casuale i pixel sbagliati, l'errore diventa meno visibile ai nostri occhi, dato che siamo portati per natura ad accorgerci di motivi regolari, più che di distorsioni casuali.
Questo concetto, che funziona meravigliosamente per le immagini, può diventare facilmente un incubo nelle animazioni. Dato che per sua natura jitter è casuale, darà risultati differenti per ogni fotogramma che renderizziamo e darà differenze ancora più evidenti se riduciamo l'immagine finale a 256 colori (ad esempio, per formati FLC o GIF animate). Il risultato sono pixel che saltellano qua e là per la scena, ma soprattutto in quei punti dove normalmente ci sarebbero problemi di aliasing (ad esempio, dove un piano infinito incontra l'orizzonte).
Per questo motivo, dovremmo sempre impostare `jitter off` quando ci prepariamo a renderizzare un'animazione. Il (relativamente) piccolo miglioramento nella qualità che possiamo ottenere, verrebbe travolto dall'oceano di pixel saltellanti che ne risulterebbe. Questa regola generale si applica anche a tutti quegli elementi di texture veramente casuali, come `crand`.

4.10.6.5 Impostazioni dei File INI

Bene, ora abbiamo una minima conoscenza di come scrivere codice rivolto alle animazioni per POV-Ray. Conosciamo la variabile `clock`, sappiamo definire variabili ausiliarie dipendenti da `clock` e sappiamo usare la parola chiave `phase`. Sappiamo che non si devono usare `jitter` o `crand` quando renderizziamo un'animazione e siamo pronti per costruirne una. Ai posti, partiamo !
La prima cosa di cui ci dobbiamo occupare sono le impostazioni dei file .INI, `Initial_Frame` e `Final_Frame`. Queste sono impostazioni molto comode che ci permettono di renderizzare un particolare numero di fotogrammi, ciascuno caratterizzato da un suo numero d'ordine, in maniera completamente automatica. E' talmente semplice che non richiederebbe spiegazioni. Ad ogni modo, qui c'è un esempio. Aggiungiamo le seguenti linee al nostro file .INI preferito.

```
Initial_Frame = 1
Final_Frame = 20
```

Ed inizieremo un loop automatico che renderizzerà 20 fotogrammi diversi. Le impostazioni stesse aggiungeranno un numero progressivo ad ogni fotogramma che viene renderizzato, qualunque sia il nome che abbiamo scelto di dargli, rendendo così unico ogni fotogramma, senza bisogno che ce ne preoccupiamo. Inoltre, per default, farà variare il valore di `clock` tra zero ed uno ad incrementi inversamente proporzionali al numero totale di fotogrammi (ad esempio, se abbiamo 10 fotogrammi, al primo corrisponderà il valore 0, al primo 1/10. Al secondo, 2/10 e così via). Questo è molto conveniente, dato che otterremo un valore di `clock` che va da 0 ad 1 indipendentemente dal fatto che stiamo renderizzando una GIF animata da 5 fotogrammi o un filmato MPEG da 300. L'animazione inizierà e terminerà esattamente nello stesso modo, ma sarà distribuita in un numero

di fotogrammi diverso.
Ora, occupiamoci di `clock`. Nel nostro esempio con la palla che rotola, abbiamo visto che talvolta possiamo avere bisogno che `clock` vari entro un intervallo diverso dal classico 0-1. Bene, quando ciò è necessario abbiamo un'impostazione anche per questo. Per fare variare `clock`, come nel nostro esempio, da 0 a 2, dobbiamo aggiungere le seguenti linee al nostro file .INI :

```
Initial_Clock = 0.0
Final_Clock = 2.0
```

Ora, supponiamo di ottenere una sequenza di 100 fotogrammi e che abbiamo notato un qualche difetto, diciamo tra i fotogrammi 51 e 75. Ritorniamo sul nostro file scena e lo correggiamo. Ora vorremmo renderizzare solo quei 25 fotogrammi, invece di dovere rifare tutta la sequenza dall'inizio. Cosa dobbiamo fare ?
Se scriviamo nel nostro file .INI le linee

```
Initial_Frame = 51
Final_Frame = 75
```

stiamo sbagliando. Anche se questo comando ci darebbe dei fotogrammi numerati tra 51 e 75, questi non potrebbero essere reinseriti nella nostra sequenza, dato che `clock` varierebbe comunque tra 0 (fotogramma 51) e ed 1 (fotogramma 75). L'unico caso in cui `Initial_Frame` e `Final_Frame` dovrebbero essere usati è quando stiamo preparando una sequenza nuova che deve venire attaccata alla fine di una già esistente.
Se vogliamo vedere solo i fotogrammi da 51 a 75 dell'animazione originale, abbiamo bisogno di due nuove impostazioni nel nostro file .INI :

```
Subset_Start_Frame = 51
Subset_End_Frame = 75
```

Che, aggiunte al file .INI, fanno variare `clock` tra i valori proporzionali a tutti e 100 i fotogrammi, ma renderizzano solo i fotogrammi da 51 a 75.
Ciò ci dovrebbe fornire un'idea elementare di come utilizzare l'animazione, anche se questo tutorial introduttivo non esaurisce tutti gli argomenti. Ad esempio, gli ultimi due parametri possono assumere valori frazionari, come 0.5 o 0.75 in modo da non cambiare il numero di fotogrammi indipendentemente da quale parte dell'animazione viene renderizzata. Inoltre c'è supporto per rendering per campi di linee pari e dispari, utile per animazioni preparate per essere riprodotte in maniera interlacciata, come in televisione (vedi il capitolo 7 per maggiori dettagli).

POV-Ray 3 supporta una grande quantità di opzioni per l'animazione, aggiungendo una quarta dimensione al raytracing. Sia che stiamo preparando un filmato FLIC, AVI, MPEG, o semplicemente una GIF animata per la nostra pagina web, il processo di animazione ci risparmia molte noie. E non dimentichiamo che `phase` e `clock` possono essere usati per esplorare e modificare molti elementi di texture ed alcuni degli oggetti più difficili da gestire (ad esempio, i frattali julia). Quindi, anche se siamo soddisfatti delle nostre scene 'ferme', aggiungere l'animazione al nostro repertorio può portarci ad una grande comprensione di ciò che POV-Ray è capace di fare. L'avventura aspetta !

5. Guida al Linguaggio di POV-Ray

Questi capitoli di riferimento descrivono tutti i parametri per la linea di comando e le parole chiave dei file .INI che vengono utilizzati per impostare le opzioni di POV-Ray. Si suppone che essi vengano usati come riferimento. Non contengono dettagliate spiegazioni su come vengono scritte le

scene, o su come si usa POV-Ray. Semplicemente, spiegano tutte le funzioni, la loro sintassi, le applicazioni, limiti ed imperfezioni ecc.

6. Opzioni di POV-Ray

POV-Ray è stato originariamente creato come programma a linea di comando per sistemi operativi senza interfaccia grafica, senza finestre di dialogo e senza menù a tendina. Molte versioni di POV-Ray usano ancora le linee di comando per specificare determinati parametri.

6.1 Impostare le opzioni di POV-Ray.

Ci sono due distinte maniere di impostare le opzioni di POV-Ray : parametri espressi per mezzo della linea di comando e opzioni da specificare nei file **.ini**. Entrambi vengono spiegati in dettaglio nelle seguenti sezioni.

6.1.1 Parametri (switches) per la linea di comando.

Questi parametri sono composti da un segno + (più) o da un - (meno), seguiti da uno o più lettere dell'alfabeto e all'occorrenza da un numero. Ecco un esempio di linea di comando :

```
POVRAY +Isimple.pov +V +W80 +H60
```

povray `è il nome del programma ed è seguito da alcuni parametri. Ogni parametro inizia con un segno di + o -. Il +I` con il nome del file dice a POV-Ray quale file deve usare come input e il `+V` dice al programma di mostrare il suo stato sullo schermo mentre sta lavorando.
Il `+W` e il `+H` impostano la larghezza (*width*) e l'altezza (*height*) dell'immagine in pixels. Questa immagine sarà larga 80 pixels e alta 60 pixels.
Per quei parametri relativi ad una funzione che può essere inserita o disinserita, il segno più la attiva e il meno la disattiva. Per esempio `+P` attiva la pausa alla fine di un rendering, mentre `-P` la disattiva. Altri parametri sono usati per specificare valori e anche qui possono comparire il più ed il meno, per esempio `+W320` imposta la larghezza di 320 pixels, ma si può anche scrivere `-W320` ed avere lo stesso risultato.
I parametri vengono letti da sinistra a destra ma in generale possono essere specificati in qualsiasi ordine. Se un parametro viene specificato più di una volta il valore precedente viene solitamente soprascritto da quello successivo. L'unica eccezione è per `+L` che imposta le librerie. Si possono specificare fino a dieci percorsi (path).
Quasi tutti i parametri hanno un'equivalente opzione che può essere usata nei file **.ini**, che sono descritti nella prossima sezione. Una dettagliata descrizione di ogni parametro si può trovare nella sezione "Guida di Riferimento alle Opzioni di POV-Ray".

6.1.2 Usare i file .INI

Essendo difficoltoso impostare diverse opzioni nelle righe di comando è possibile inserire una o più opzioni in un file di testo. Questi file hanno estensione **.ini** (file di inizializzazione). Precedenti versioni di POV-Ray li chiamavano file .DEF (default). E' ancora possibile utilizzare file .DEF esistenti con questa versione di POV-Ray.
La maggioranza delle opzioni che si usano possono venire raccolte in un file**.ini**. le righe di comando sono consigliabili per opzioni che vengono attivate e disattivate spesso nel momento in cui viene renderizzata una scena alla quale si sta lavorando. Il file **povray.ini** è letto automaticamente se è presente. E' possibile specificare ulteriori file **.ini** nelle righe di comando semplicemente digitando il nome del file. Per esempio :

```
POVRAY MYOPTS.INI
```

Se non viene specificata alcuna estensione, allora si assume che sia **.ini**. POV-Ray riconosce che `MYOPTS.INI` non è un parametro perché non è preceduto da un più o da un meno. Infatti un errore comune in coloro che utilizzano da poco POV-Ray è quello di dimenticarsi di mettere il parametro `+I` davanti al nome del file di input. Senza il parametro POV-Ray "crede" che il file **simple.pov** sia un file **.ini**. Non dimenticate ! Se nessun più o meno precede un parametro della linea di comando, quel parametro viene considerato come una specificazione di file **.ini**.
Nella riga di comando possono essere presenti più file **.ini** e anche parametri. Per esempio :

```
POVRAY MYOPTS +V OTHER
```

Questa linea di comando legge le impostazioni dal file **myopts.ini**, imposta il parametro `+V` e legge le impostazioni dal file **other.ini**.

Un file **.ini** è un normale file di testo ASCII, con opzioni specificate in una sintassi del tipo...

```
Parola_dell'opzione=VALORE ; Il testo dopo il punto e virgola è un
commento.
```

per esempio il file **.ini** equivalente al parametro `+Isimple.pov` è...

```
Input_File_Name=simple.pov
```

Le opzioni sono lette nel file dall'alto verso il basso, ma possono essere specificate in qualsiasi ordine. Se si specifica più di una volta la stessa opzione, il valore della precedente è generalmente sovrascritto dall'ultima. L'unica eccezione è l'opzione

```
Library_Path=percorso
```

Si possono specificare fino a dieci percorsi (path).
Quasi tutte le opzioni dei file **.ini** hanno un equivalente nei parametri della linea di comando. La sezione "Guida di Riferimento alle Opzioni di POV-Ray " dà una dettagliata descrizione di tutte le opzioni di POV-Ray, incluse sia quelle dei file **.ini** che quelle dei parametri.
Le parole chiave dei file **.ini** non sono sensibili alle maiuscole. E' consentito usare solo un'opzione per riga di testo. E' possibile includere anche parametri della riga di comado nei file **.ini**, se ciò risulta di aiuto. Si possono scrivere più parametri per riga, ma senza mescolarli nella stessa riga con le opzioni dei file **.ini**. Si possono "annidare" (inserire l'uno nell'altro) i file **.ini** semplicemente mettendo il nome del file in una riga da solo e senza segno di uguale in fondo. Si possono annidare file **.ini** fino a dieci livelli.
Per esempio :

```
; Questo è un file .ini di esempio, tutta questa linea è di
commento
;Sono permesse le linee vuote

Input_File_Name=semplice.pov
;Questa linea imposta il nome del file di input

+W80 +H60
; Sono permessi anche i comandi con + e -
```

```
MOREOPT
; Leggi il file MOREOPT.INI e continua con la prossima linea

+V
; Un altro parametro
; E questo è tutto !
```

I file **.ini** possono avere diverse sezioni alle quali può venire assegnata un'etichetta che deve essere posta tra parentesi quadre. Per esempio :

```
; RES.INI
; Questo file .ini di esempio è usato per impostare la risoluzione

+W120 +H100 ; Questa sezione non ha etichetta.
; Selezionala col comando "RES"

[Bassa]
+W80 +H60 ; Questa sezione ha un'etichetta.
; Selezionala col comando "RES[Bassa]"

[Media]
+W320 +H200 ; Questa sezione ha un'etichetta.
; Selezionala col comando "RES[Media]"

[Alta]
+W640 +H480 ; Le etichette non sono sensibili alle maiuscole.
; "RES[alta]" funziona ugualmente

[Molto Alta]
+W800 +H600 ; Le etichette possono contenere spazi.
```

Quando si specifica un file **.ini** si deve aggiungere il nome dell'eventuale etichetta tra parentesi quadre. Ad esempio :

```
POVRAY RES[Media] +Imyfile.pov
```

POV-Ray legge **res.ini** e ignora tutte le opzioni finchè non trova l'etichetta Media ed esegue le opzioni specificate da questa etichetta finché non ne trova un'altra, dopo di che ignora il resto. Se nessuna etichetta è specificata nella linea di comando, viene letta solo l'area senza etichetta in cima al file. Se un'etichetta è specificata, allora l'area senza etichetta viene ignorata.

6.1.3 Usare la Variabile d'Ambiente POVINI.

La variabile d'ambiente POVINI è utilizzata per specificare la posizione e il nome di un file **.ini** di default che viene letto ogni volta che si esegue POV-Ray. Se POVINI non è specificata può essere letto un file **.ini** di default a seconda della piattaforma utilizzata. Se il file specificato non esiste viene visualizzato un messaggio di avvertimento (*warning*).
Per impostare le variabili d'ambiente sotto MS-DOS è necessario inserire nel file **autoexec.bat** la seguente riga :

```
set POVINI=c:\povray3\default.ini
```

Su quasi tutti i sistemi operativi la sequenza di lettura delle opzioni è la seguente :

1. legge opzioni dal file **.ini** di default specificato dalla variabile d'ambiente POVINI o da uno specifico file **.ini**
2. legge i parametri dalla riga di comando (questo include leggere ogni file **.ini** o **.def** lì specificato).

La variabile d'ambiente POVRAYOPT supportata dalle precedenti versioni di POV-Ray non è più disponibile.

6.2 Guida di Riferimento alle Opzioni di POV-Ray.

Come spiegato nella precedente sezione, le opzioni devono essere specificate da parametri nella linea di comando o con le opzioni dei file **.ini**. Quasi tutti le opzioni dei file **.ini** hanno un equivalente nei parametri e la maggioranza dei parametri ha equivalenti nelle opzioni dei file **.ini**. La sezione che segue dà una dettagliata descrizione di ogni opzione di POV-Ray e include sia le opzioni dei file **.ini** che quelle dei parametri.
La notazione e la terminologia usata sono descritte nella tabella a seguito.

`Parola_chiave=bool`	attiva la funzione se `bool` è uguale a `true`, `yes`, `on` o `1` e la disattiva se c'è qualunque altro valore.
`Parola_chiave=true`	utilizzare questa opzione se è specificato `true`, `yes`, `on` o `1`.
`Parola_chiave =false`	utilizzare questa opzione se è specificato false, no, off or 0
`Parola_chiave =file`	qualunque nome di file valido. **Nota** : alcune opzioni proibiscono l'uso di alcune delle precedenti espressioni quali true o false come nomi di file e sono annotate nelle successive sezioni.
N	qualunque numero intero come +W320
`n.n`	qualunque numero decimale come Clock=3.45
`0.n`	qualunque numero decimale minore di 1.0 anche se non ha lo 0 iniziale
`s`	qualunque stringa di testo
`x o y`	qualunque carattere singolo
`path`	qualunque nome di directory, con a scelta il drive, senza barra finale (\ o / a seconda del sistema operativo)

Senza ulteriori specificazioni, si può assumere che un segno più o meno davanti ad un parametro produca lo stesso risultato.

6.2.1 Opzioni sull'Animazione

POV-Ray 3.0 ha molto migliorato le sue capacità in fatto di animazione introducendo la possibilità di un *loop interno di animazione*, di passare a programmi di utilità esterni che possono assemblare singoli fotogrammi e creare l'animazione e la numerazione automatica del nome dei file di output. Il loop interno che consente l'animazione è semplice e flessibile, anche se è sempre possibile utilizzare programmi esterni o file batch per creare le animazioni come nella versione 2.

6.2.1.1 Loop esterno di animazione.

```
Clock=n.n  Imposta "clock" come variabile al valore n.n
+Kn.n lo stesso di Clock=n.n
```

L'opzione `clock=n.n` o il parametro `+Kn.n` possono essere usati per fornire al programma una singola variabile per animazioni di base. Il valore è assegnato alla variabile `clock`. Se un oggetto deve ruotare in questo modo <0,clock,0> allora è possibile ruotarlo di quantità diverse su fotogrammi diversi impostando +K10.0, +K20.0...ecc. nei rendering successivi. E' a discrezione dell'utente dare diversi valori di clock e diversi nomi per i file di output.

6.2.1.2 Loop Interno di Animazione.

`Initial_Frame=n`	Imposta il numero di fotogramma iniziale a `n`.
`Final_Frame=n`	Imposta il numero dell'ultimo fotogramma.
`Initial_Clock=n.n`	Imposta il numero di clock inziale.
`Final_Clock=n.n`	Imposta il numero di clock finale.
`+KFIn`	Lo stesso che `Initial_Frame=n`
`+KFFn`	Lo stesso che `Final_Frame=n`
`+KIn.n`	Lo stesso che `Initial_Clock=n.n`
`+KFn.n`	Lo stesso che `Final_Clock=n.n`

La novità del loop interno per le animazioni nella versione di POV-Ray 3.0 permette di non dover creare complicate serie di file batch per avviare più volte POV-Ray con impostazioni differenti. Anche se la molteplicità delle opzioni può intimidire, per l'intelligente impostazione dei valori di default, basterà, nella maggior parte dei casi, impostare solo `Final_Frame=n` o `+KFFn` per specificare il numero dei fotogrammi. Tutti gli altri valori possono rimanere come quelli predefiniti. Ogni impostazione di `Final_Frame` diversa da -1 farà partire il loop interno di POV-Ray. Per esempio

`Final_Frame=10` oppure `+KFF10`

permette a POV-Ray di renderizzare la tua scena 10 volte. Se viene specificato `output_File_Name=`**`file.tga`** allora per ogni fotogramma si avrà un file di output che si chiamerà **file01.tga**, **file02.tga**, **file03.tga** ecc. Il numero di zeri dipende dal numero totale dei fotogrammi. Per esempio `+KFF100` genererà da **file001.tga** a **file100.tga**. Il numero progressivo del fotogramma può 'invadere' il nome del file. Con MS-DOS ed il suo limite di 8 caratteri per il nome, **myscene.pov** darebbe 100 fotogrammi da **mysce001.tga** a **mysce100.tga**.
Il valore di default di `Initial_Frame=1` non avrà probabilmente mai bisogno di essere cambiato. Si potrebbe aver bisogno di cambiarlo solo se si stanno assemblando insieme parti di complesse animazioni divise in distinte sequenze, allora una scena potrà andare dal fotogramma 1 al 50 e la successiva dal 51 al 100 ; e proprio a questo servono i parametri

`Initial_Frame=n e +KFIn.`

Nota che se volessi renderizzare un sottoinsieme di fotogrammi, come ad esempio i fotogrammi da 30 a 40 di un'animazione che va dal fotogramma 1 al fotogramma 100, non dovresti cambiare i valori di `Initial_Frame` o `Final_Frame`. Dovresti invece usare i comandi descritti nel paragrafo "Sottoinsiemi di fotogrammi".
Diversamente da altri programmi di animazione, l'azione nelle scene animate di POV-Ray non

dipende solo dal numero progressivo del fotogramma. Infatti si organizza la scena grazie alla variabile `clock` . Di default il valore di `clock` è 0.0 per il fotogramma iniziale e di 1.0 per quello finale mentre tutti gli altri sono interpolati tra questi due valori. Per esempio se il tuo oggetto dovesse ruotare facendo un giro completo durante il corso dell'animazione, è possibile specificare `rotate 360*clock*y`, cosicché via via che la variabile `clock` varia tra 0.0 e 1.0, l'oggetto ruota rispetto all'asse y da 0 a 360 gradi.
Il maggiore vantaggio di questo sistema è che si possono renderizzare animazioni con 10, 100, 500 o 329 fotogrammi e l'oggetto ruoterà comunque di 360 gradi tra il primo e l'ultimo fotogramma : animazioni di prova con pochi fotogrammi lavoreranno quindi nello stesso modo dei rendering finali di animazioni molto complesse.
In effetti il movimento è definito con continuità al variare di un parametro e vengono presi campioni discreti del movimento a intervalli fissi, cioè i fotogrammi. Un film o una registrazione con una telecamera di una scena reale funziona nello stesso modo. Il movimento di un oggetto dipende dal passare del tempo, non dalla velocità dei fotogrammi.
Molti fra coloro i quali usano POV-Ray hanno già creato animazioni che vanno oltre i valori di clock da 0.0 a 1.0. Ed è questa la ragione per cui si è pensato di introdurre i parametri `Initial_Clock=n.n` o `+KIn.n` e `Final_Clock=n.n` o `+KFn.n`. Per esempio, per far scorrere il `Clock` da 25.0 a 75.0 bisognerà specificare `Initial_Clock=25.0` e `Final_Clock=75.0` ed allora il clock sarà impostato a 25.0 per il fotogramma iniziale e a 75.0 per quello finale, nel mezzo i fotogrammi avranno valori di clock interpolati tra 25.0 e 75.0.
Coloro che sono abituati a usare il numero di fotogrammi piuttosto che i valori di clock possono specificare

`Initial_Clock=1.0` e `Final_Clock=10.0` e `Frame_Final=10`

per ottenere un'animazione con dieci fotogrammi.
Per le nuove scene, si raccomanda di non cambiare `Initial_Clock` o `Final_Clock` dai loro valori di default di 0.0 e 1.0, se si vuole far variare il clock su un intervallo diverso da 0.0-1.0, è preferibile aggiungere le seguenti righe al file della scena

```
#declare Start = 25.0
#declare End = 75.0
#declare My_Clock = Start+(End-Start)*clock
```

ed usare poi `My_Clock` nella descrizione della scena. Questo mantiene i valori di 25.0 e 75.0 nel file **.pov**.
Ulteriori dettagli riguardanti le impostazioni per il loop sono nella sezione sui comandi per l'uscita al sistema operativo nella sezione "Uscita al Sistema Operativo".

6.2.1.3 Sottoinsiemi di Fotogrammi

`Subset_Start_Frame=n`	Imposta il fotogramma iniziale del sottoinsieme da n
`Subset_Start_Frame=0.n`	Imposta il fotogramma iniziale del sottoinsieme da n%
`Subset_End_Frame=n`	Imposta il fotogramma finale del sottoinsieme a n
`Subset_End_Frame=0.n`	Imposta il fotogramma finale del sottoinsieme a n%
`+SFn o +SF0.n`	Lo stesso che `Subset_Start_Frame`
`+EFn o +EF0.n`	Lo stesso che `Subset_End_Frame`

Quando si crea una lunga animazione può essere più maneggevole renderizzare solo una parte di questa per vedere come risulta. Supponendo di avere 100 fotogrammi ma volendo renderizzare solo dal 30 al 40, se si impostano le opzioni

`Initial_Frame=30` e `Final_Frame=40`

il clock varierà da 0.0 a 1.0 dal fotogramma 30 al 40, invece di variare tra 0.30 e 0.40 come dovrebbe. Basta quindi lasciare invariati

`Initial_Frame=1` e `Final_Frame=100`

e usare anche

`Subset_Start_Frame=30` e `Subset_End_Frame=40`

per renderizzare solo *quella* parte di animazione con i giusti valori di clock.
Solitamente si specifica il sottoinsieme usando il numero di fotogrammi utilizzato, ma un'alternativa può essere di prendere un numero decimale compreso tra zero ed uno, che viene interpretato come una frazione di tutta l'animazione. Per esempio,

`Subset_Start_Frame=0.333` e `Subset_End_Frame=0.667`

renderizzerà il terzo centrale di un'animazione indipendentemente dal numero di fotogrammi.

6.2.1.4 Animazioni Cicliche

`Cyclic_Animation=bool`	Attiva/disattiva l'opzione di animazione ciclica (on/off)
`+KC`	Attiva l'opzione di animazione ciclica
`-KC`	Disattiva l'opzione di animazione ciclica

Molte sequenze animate realizzate con il computer sono congegnate per ripetersi lungo un anello. Supponendo di avere un oggetto che debba ruotare esattamente 360° lungo l'intero arco di un'animazione e si sia scritto il comando `rotate 360*clock*y` a questo scopo, sia il primo che l'ultimo fotogramma saranno identici, quindi nell'animazione ci sarà un momento di 'salto'. Per eliminare questo inconveniente è necessario aggiustare il clock cosicché l'ultimo fotogramma non sia identico al primo, ma in sequenza con esso. Per esempio un'animazione di 10 fotogrammi dovrebbe usare il clock da 0.0 a 1.0, ma in questo caso `clock` dovrebbe essere cambiato e variare da 0.0 a 0.9 con incrementi di 0.1. Se invece si cambia il clock in un'animazione di 20 fotogrammi, allora dovrebbe passare da 0.0 a 0.95 con incrementi di 0.05. Tutto questo è complicato ancora di più dal fatto che sarebbe necessario cambiare il valore finale del clock ogni qual volta si dovesse cambiare il parametro `Final_Frame`. Impostando `Cyclic_Animation=on` o usando `+KC` POV-Ray aggiusterà automaticamente il valore finale del clock in relazione al numero totale dei fotogrammi. Il valore di default di queste impostazioni è `off`.

6.2.1.5 Rendering per campi

`Field_Render=bool`	Attiva/disattiva l'opzione di rendering per campi (on/off)
`Odd_Field=bool`	Attiva l'opzione per il rendering dei campi dispari (on/off)
`+UF`	Attiva il rendering per campi
`-UF`	Disattiva il rendering per campi

+UO	Attiva il rendering dei campi dispari
-UO	Disattiva il rendering dei campi dispari

Il rendering per campi è solitamente usato quando l'animazione verrà trasmessa via televisione. La televisione mostra, per ogni refresh verticale dello schermo, una passata di linee orizzontali alternate. Ogni volta che un fotogramma viene mostrato i due campi di linee alternate sono interlacciati per dare l'impressione di una maggiore risoluzione dell'immagine. Le linee pari formano il campo pari e sono tracciate per prime (cioè le linee 0, 2, 4, ecc.), seguite dal campo dispari, formato dalle linee dispari che vengono tracciate subito dopo. Se gli oggetti in un'animazione si muovono velocemente, la loro posizione può cambiare notevolmente da un campo a quello successivo. Potrà quindi essere preferibile renderizzare a campi alternati alla velocità effettiva dei campi (che è il doppio della velocità dei fotogrammi),piuttosto che renderizzare interi fotogrammi alla velocità normale.
Di default il rendering per campi non è usato. Impostando `Field_Render=on` o usando `+UF` si farà in modo che i fotogrammi di un'animazione siano solo i campi pari e dispari alternativamente. Per default il primo fotogramma è il campo pari, seguito dal campo dispari. Puoi far renderizzare a POV-Ray il campo dispari per primo specificando `Odd_Field=on`, o usando il parametro `+UO`.

6.2.2 Opzioni di Output

6.2.2.1 Opzioni Generiche di Output.

6.2.2.1.1 Altezza e Larghezza dell'Output

Height=n	Imposta l'altezza uguale a `n` pixels.
Width=n	Imposta la larghezza uguale a `n` pixels.
+Hn	Lo stesso di `Height=n` (quando n > 8)
+Wn	Lo stesso di `Width=n`

Questi parametri impostano l'altezza e la larghezza del file di output in pixels. La finestra di anteprima, se attivata, cercherà in generale di avere una modalità video compatibile con queste misure, ma in ogni caso le impostazioni del display non influenzeranno l'immagine risultante.

6.2.2.1.2 Opzioni di Output Parziale.

Start_Column=n	imposta la prima colonna a n
Start_Column=0.n	imposta la prima colonna a n% della larghezza
+SCn o +SC0.n	lo stesso di `Start_Column`
Start_Row=n	imposta la prima riga a n pixels
Start_Row=0.n	imposta la prima riga a n% dell'altezza
+SRn o +Sn	lo stesso di `Start_Row=n`
+SR0.n o +S0.n	lo stesso di `Start_Row=0.n`
End_Column=n	imposta l'ultima colona a n pixels
End_Column=0.n	imposta l'ultima colonna a n% della larghezza

+ECn o +EC0.n	lo stesso di End_Column
End_Row=n	imposta l'ultima riga a n pixels
End_Row=0.n	imposta l'ultima riga a n% dell'altezza
+ERn o +En	lo stesso di End_Row=n
+ER0.n o +E0.n	lo stesso di End_Row=0.n.

Quando si fanno dei rendering di prova è spesso conveniente renderizzare solo una piccola regione di spazio rispetto all'intera immagine per controllare velocemente alcuni particolari. Le opzioni di Start_Row, End_Row, Start_Column e End_Column permettono di definire porzioni di area da renderizzare. Il valore di default è l'intera immagine da (1,1) in alto a sinistra a (w,h) in basso a destra, dove con w si intende il valore assegnato alla larghezza Width=n e con h si intende il valore assegnato all'altezza Height=n.
E' da notare che se il numero assegnato è maggiore di 1 allora viene interpretato come il numero assoluto di pixels in una riga o in una colonna, se invece è un numero decimale compreso tra 0.0 e 1.0 allora è interpretato come una percentuale del totale dell'altezza o della larghezza dell'immagine. Per esempio : Start_Row=0.75 e Start_Column=0.75 inizia il rendering da una riga posta al 75% (dall'alto verso il basso) dell'immagine, e da una colonna al 75% (dalla sinistra). Viene quindi renderizzato solo il 25% dell'immagine che si trova in basso a destra.
I parametri +SR, +ER, +SC e +EC fanno la stessa cosa dei i corrispondenti comandi per i file **.ini**, le prime versioni di POV-Ray supportavano solamente i comandi per le righe iniziale e finale +Sn e +En.

6.2.2.1.3 Opzioni di Interruzione del Rendering

Test_Abort=bool	Attiva/disattiva la possibilità di interruzione del rendering (on/off)
+X	Attiva l'interruzione
-X	Disattiva l'interruzione
Test_Abort_Count=n	Attiva la possibilità di interruzione del rendering ogni n pixels
+Xn	Attiva la possibilità di interrompere ogni n pixels
-Xn	Disattiva la possibilità di interrompere ogni n pixels (in futuro controlla ogni n pixels)

Su alcuni sistemi operativi quando si avvia un rendering è necessario lasciarlo terminare. L'opzione Test_Abort=on o il parametro +X permette di interrompere il rendering tramite la pressione di un tasto : il file verrà fermato e saranno salvate solo le righe interamente completate. POV-Ray esce da questo rendering con un codice di errore 2 (normalmente per un rendering finito il codice è 0 e per un errore fatale è 1).
Quando è attivata questa opzione la tastiera è controllata una volta per ogni linea mentre viene eseguito il parsing e una per ogni pixel mentre viene eseguito il rendering. Poiché il controllo della tastiera può rallentare il rendering, l'opzione Test_Abort_Count=n o il parametro +Xn fanno sì che il test venga eseguito solo ogni n pixels della scena renderizzata o ogni n righe del file della scena durante il parsing.

6.2.2.1.4 Opzioni di Continuazione del Rendering

Continue_Trace=bool	Imposta la continuazione di un rendering interrotto.
+C	attiva la continuazione di un rendering interrotto.
-C	disattiva la continuazione di un rendering interrotto.
Create_Ini=file	genera un file .ini di nome file.ini
Create_Ini=true	genera un file .ini con lo stesso nome del file scena
Create_Ini=false	Disattiva l'uso di un precedente file.ini
+GIsss	Lo stesso che Create_Ini=sss

Se si interrompe un rendering mentre è in corso, o si usa l'opzione End_Row per interromperlo prima della fine, è possibile usare queste opzioni per continuare più tardi il rendering, riprendendolo dal punto in cui era stato interrotto. L'opzione legge il file di output precedentemente generato, eventualmente mostra la parte di immagine renderizzata fino ad allora e procede con il rendering del resto. Questa opzione non può essere usata se l'output su file è disattivato con

Output_to_file=off o col parametro -F.

L'opzione Continue_Trace potrebbe non funzionare se l'opzione Start_Row è stata impostata a partire da una riga di inizio diversa dalla prima, in relazione al formato di output usato. POV-Ray tenta di riprendere un rendering interrotto leggendo tutte le informazioni disponibili su quel file nello specifico file di output. Tutti i formati dei file contengono le misure dell'immagine e questo sovrascriverà ogni altra dimensione specificata. Alcuni formati di file come ad esempio .tga e .png immagazzinano informazioni anche sul punto di inizio del file (+SCn e +SRn), sull'output del canale alfa +UA e sul numero di colori usati +FNn, che quindi sovrascriveranno queste impostazioni. E' compito dell'utente accertarsi che tutte le altre opzioni siano impostate come nel rendering originale.
L'opzione Create_Ini o il parametro +GI permettono di creare un file **.ini** con tutte le impostazioni usate per il rendering, così è possibile far ripartire il file con le stesse opzioni o assicurarsi che quelle attuali siano le stesse. Questa opzione crea un file **.ini** con tutte le impostazioni usate per quel rendering e include i valori di default che non sono stati specificati. Per esempio se si avvia POV-Ray con

```
POVRAY +Isimple.pov MYOPTS +GIrerun.ini MOREOPTS
```

POV-Ray creerà un file chiamato **rerun.ini** con tutte le opzioni usate per generare la scena. Il file non verrà scritto finché tutte le opzioni non saranno state analizzate. questo significa che nell'esempio qui sopra il file includerà le opzioni sia del **myopts.ini** che del **moreopts.ini** anche se il +g è specificato fra il nome dei due file. E' ora possibile riavviare la scena con il file...

```
POVRAY RERUN
```

o ricominciare una scena interrotta con

```
POVRAY RERUN +C
```

se si aggiungono altri parametri con le opzioni di **rerun.ini** questi saranno inclusi nei riavvii futuri perché il file sarà riscritto ogni volta che viene usato.
L'opzione Create_Ini è anche utile per documentare come una scena è stata renderizzata. Se si

renderizza **waycool.pov** con Create_Ini=on allora sarà creato un file **waycool.ini** che può essere distribuito con il file contenente la scena in modo che altri utenti possano renderizzare la scena allo stesso modo.

6.2.2.2 Opzioni di Output a Schermo

6.2.2.2.1 Impostazioni Hardware

Display=bool	Attiva /disattiva la visualizzazione a schermo durante il rendering (on/off)
+D	attiva la visualizzazione a schermo durante il rendering Turns graphic display on
-D	disattiva la visualizzazione a schermo durante il rendering
Video_Mode=x	Imposta la modalità video a x, non si devono usare i comandi on/off
+Dx	attiva la visualizzazione a schermo durante il rendering nella modalità video x Attiva la
-Dx	disattiva la visualizzazione a schermo durante il rendering; in futuro imposterà l'uso della modalità x
Palette=y	imposta la tavolozza colori del video a y; non è influenzato da on/off
+Dxy	Attiva la visualizzazione durante il rendering con modalità x e paletta y
-Dxy	Disattiva la visualizzazione durante il rendering ; in futuro verrà usata modalità x e tavolozza colori y
Display_Gamma=n.n	Imposta la gamma colore a n

Imposta lo schermo gamma a n.n
L'impostazione Display=on o il parametro +D permette di mostrare l'immagine mentre viene renderizzata, anche se alcuni sistemi non hanno uscita grafica a schermo POV-Ray è in grado di mostrare una versione dell'immagine 80 per 24 con caratteri ASCII-ART. Nei casi in cui sia possibile, l'immagine sarà visualizzata a schermo intero a 24 bit. Impostando Display=off o usando il parametro -D verrà disattivata la possibilità di vedere l'immagine mentre viene renderizzata, opzione che è invece impostata per default.

L'opzione Video_Mode=x imposta la modalità grafica o il tipo di hardware scelto, dove x è una singola cifra o lettera che è dipendente dalla macchina (vedere sezione "Tipi di Display " per una descrizione delle modalità supportate da MS-DOS). Generalmente Video_Mode=0 significa l'impostazione di default o l'impostazione rilevata automaticamente. Quando si usano i parametri la modalità è indicata allo stesso modo con un numero che segue immediatamente il parametro stesso, per esempio +D0 attiverà la modalità di default.

L'opzione Palette=y seleziona la tavolozza colori (*palette*) che verrà usata. Generalmente y sarà una cifra che seleziona una specifica palette, oppure una lettera quale G per il display a toni di grigio, H per high color a 15 o 16 bit, T per true color (24 bit). Quando si usano i parametri, il carattere che definisce la palette occupa la seconda posizione, per esempio +D0T permetterà di mostrare il rendering a schermo durante la sua esecuzione, con la modalità grafica predefinita e in true color.

L'opzione `Display_Gamma=n.n` è nuova nella versione POV-Ray 3.0 e non è disponibile come parametro. Questa opzione cerca di risolvere il problema di immagini (fatte con il raytracing o no) che hanno una diversa luminosità quando vengono mostrati su monitor diversi, con diverse schede video e diversi sistemi operativi. Questa impostazione si basa sull'hardware grafico e dovrebbe essere impostata correttamente *una volta* e non più cambiata. L'impostazione `Display_Gamma` nel file `.INI` lavorando congiuntamente alle nuove impostazioni `assumed_gamma` permette a POV-Ray di creare immagini uguali su ogni sistema. Vedere la sezione "Gamma Colore" che descrive l'opzione `assumed_gamma` delle impostazioni generali più dettagliatamente.
Anche se l'opzione `Display_Gamma` può essere diversa per ogni sistema, ci sono poche regole generali per tentare di aggiustare l'opzione anche se non si è esperti. Se `Display_Gamma` non compare nelle opzione del file **.ini**, viene assunto il valore 2.2. Questo perché molti monitor dei PC hanno un valore per gamma in un raggio che va da 1.6 a 2.6 (modelli più nuovi sembra che abbiano valori per gamma più bassi). MacOS può correggere il valore di gamma (basandosi su un'impostazione dell'utente nel pannello di controllo di gamma). Se il pannello di controllo di gamma non è attivo il valore per gamma predefinito per i sistemi Macintosh è 1.8. alcune schede video più avanzate possono fare correzioni del valore di gamma via hardware e possono utilizzare il valore di `Display_Gamma` corrente, di solito 1.0. E' anche disponibile un'immagine che possa fare da test per l'impostazione corretta del `Display_Gamma`.
Per file di scena che non contengono nelle impostazioni generali `assumed_gamma` l'opzione del file **.ini** `Display_Gamma` non avrà alcun effetto sull'output di POV-Ray per quasi tutti i formati di output. Comunque il valore Display_Gamma è usato quando si creano file di output in formato PNG e anche quando si renderizzano i file di esempio di POV-Ray (perché hanno il valore di gamma impostato), cosicché possa assicurare corretti risultati per qualunque sistema.

6.2.2.2.2 Impostazioni Connesse al Display

`Pause_When_Done=bool`	Imposta la pausa quando è finito il rendering (on/off)
`+P`	Imposta la pausa quando è finito il rendering
`-P`	Disattiva la pausa quando è finito il rendering
`Verbose=bool`	visualizza i messaggi (on/off)
`+V`	visualizza i messaggi
`-V`	non visualizza i messaggi
`Draw_Vistas=bool`	visualizza i vista buffer (on/off)
`+UD`	Attiva la visualizzazione dei vista buffer
`-UD`	Disattiva la visualizzazione dei vista buffer

Su alcuni computer, quando l'immagine è stata completata, POV-Ray chiude la finestra che la contiene e torna in modalità testo per mostrare le statistiche finali ed uscire. Normalmente quando viene visualizzata l'immagine la si vuole guardare per un po' prima di continuare. Usare il comando `Pause_When_Done=on` o il parametro `+P` fa sì che POV-Ray si fermi in modalità grafica in attesa della pressione di un tasto. L'impostazione predefinita non permette la pausa a fine rendering.

Quando non si usa la modalità grafica è spesso preferibile tenere sotto controllo il procedere del rendering. Usare il comando `Verbose=on` o `+V` attiva un rapporto dettagliato sul procedere del

rendering. Riporta infatti il numero della riga che viene renderizzata, il tempo impiegato per il fotogramma corrente e altre informazioni. Su alcuni sistemi queste informazioni di testo possono causare un conflitto con la modalità grafica. Può essere necessario disattivarle quando è attiva l'opzione di visualizzazione dell'immagine. L'impostazione predefinita è `off (-V)`.

L'opzione `Draw_Vistas=on` o `+UD` era in origine una funzione per la correzione della funzione *vista buffer* di POV-Ray, ma era così divertente che abbiamo deciso di tenerla. Il vista buffering è un metodo di suddivisione dello spazio che proietta altezza e larghezza dei parallelepipedi che delimitano lo spazio tridimensionale all'interno del quale si trovano gli oggetti (*bounding boxes*) sul piano visuale dell'osservatore. POV-Ray controlla la posizione x, y di queste aree rettangolari per determinare velocemente quali oggetti saranno visibili. L'impostazione predefinita è `off` perché la rappresentazione dei rettangoli può richiedere molto tempo in scene complesse e non ha particolare utilità. Vedi il paragrafo "Controllo sul Bounding Automatico" per maggiori dettagli.

6.2.2.2.3 Anteprima a Mosaico

`Preview_Start_Size=n`	Imposta la dimensione di partenza dell'anteprima a mosaico a n pixels
`+SPn`	Lo stesso di `Preview_Start_Size=n`
`Preview_End_Size=n`	Imposta la dimensione finale dell'anteprima a mosaico a n pixels
`+Epn`	Lo stesso di `Preview_End_Size=n`

Normalmente, mentre si sta sviluppando una scena, si fanno molti rendering di prova a bassa risoluzione per vedere se gli oggetti sono posizionati correttamente. Spesso la bassa risoluzione non dà sufficiente dettaglio e quindi devi renderizzare la scena ad una risoluzione più alta. Una funzione chiamata anteprima a mosaico (*mosaic preview*) risolve questo problema renderizzando automaticamente l'immagine in fasi successive. I primi passaggi forniscono una grossolana rappresentazione dell'intera immagine usando grossi blocchi di pixel che assomigliano a piastrelle di un mosaico. L'immagine è poi raffinata usando risoluzioni più alte ai passaggi successivi. Questo metodo di visualizzazione mostra molto velocemente l'intera immagine a bassa risoluzione permettendoti di controllare i problemi di maggiore entità della scena. Man mano che l'immagine viene migliorata puoi concentrare la tua attenzione sui dettagli come ombre e texture. Non c'è bisogno di attendere un rendering a risoluzione piena per trovare i problemi dato che il rendering può essere interrotto in qualsiasi momento per correggere la scena o, se le cose vanno bene, può essere fatto proseguire per renderizzare la scena ad alta qualità e risoluzione.

Per usare questa funzione si devono prima selezionare i valori di altezza e di larghezza dell'immagine finale. L'anteprima a mosaico viene inserita specificando quanto grandi devono essere le 'tessere' del mosaico al primo passaggio, usando `Preview_Start_Size=n` o `+SPn`. Il numero n deve essere maggiore di zero ed uguale ad una potenza di 2 (1, 2, 4, 8, 16, 32, ecc.). Se non è una potenza di 2 gli viene sostituita la potenza di 2 immediatamente minore. Questo imposta la dimensione dei quadrati in pixel. Un valore di 16 disegnerà un punto ogni 16 con un quadrato di 16x16 punti al primo passaggio. I passaggi successivi useranno la metà del valore precedente (8x8, 4x4 e così via).

Il processo prosegue fino a raggiungere una dimensione di 1x1 punti o fino a raggiungere la dimensione impostata col parametro `Preview_End_Size=n` o `+EPn`. Di nuovo il valore di n deve essere un numero maggiore di zero, uguale ad una potenza di 2 e minore o uguale a `Preview_Start_Size`. Se non è una potenza di 2 viene sostituita la potenza di 2

immediatamente minore. Il valore predefinito per `Preview_End_Size` è 1. Se viene impostato ad un valore minore di 1 i passaggi termineranno prima di raggiungere la dimensione 1x1 ma POV-Ray terminerà comunque con un rendering a piena risoluzione. Per esempio se vuoi un singolo passaggio 8x8 prima di renderizzare l'immagine definitiva, imposta `Preview_Start_Size=8` e `Preview_End_Size=8`.
L'immagine non viene salvata su file fino a quando non si sia raggiunto il passaggio 1x1. Sebbene i passaggi preliminari renderizzino solo i punti necessari il passaggio 1x1 renderizza nuovamente ogni pixel in modo che l'antialiasing ed il salvataggio dell'immagine funzionino correttamente. Ciò causa una maggiore durata del rendering (fino al 25% in più) e quindi si suggerisce di non usare l'anteprima a mosaico per il rendering finale. Inoltre poiché il file non viene salvato fino all'ultimo passaggio i rendering interrotti prima non possono venire recuperati senza ricominciare dall'inizio. Future versioni di POV-Ray includeranno file temporanei che elimineranno queste inefficienze. L'anteprima a mosaico resta comunque una utile opzione per fare rendering di prova.

6.2.2.3 Opzioni di Output su File

`Output_to_File=bool`	Permette l'output dell'immagine su file (on/off)
`+F`	Permette l'output dell'immagine su file (usando il formato predefinito)
`-F`	Non permette l'output dell'immagine su file

POV-Ray è impostato in modo da permettere l'output dell'immagine su file. Quando si stanno creando delle scene e quindi facendo rendering di prova, può bastare la modalità grafica. Per risparmiare tempo e spazio su disco si può impostare `Output_to_File=off` o `-F`.

6.2.2.3.1 Tipo di File di Output

`Output_File_Type=x`	Imposta il formato del file di output a 'x'
`+Fxn`	Inserisce l'output dell'immagine su file, imposta il formato del file di output a 'x' e il numero di colori a 'n'
`-Fxn`	Disinserisce l'output dell'immagine su file, ma in futuro imposterà il formato del file di output a 'x' e il numero di colori a 'n'
`Output_Alpha=bool`	Attiva/disattiva l'output del canale alfa (on/off)
`+UA`	attiva l'output del canale alfa
`-UA`	disattiva l'output del canale alfa
`Bits_Per_Color=n`	Imposta il numero di bit per punto del file di output a 'n'

Il tipo predefinito del file immagine dipende dalla piattaforma che stai usando. MS-DOS e la maggior parte delle altre hanno output predefinito a immagini Targa a 24 bit non compresse. Controlla la documentazione specifica per la tua piattaforma per vedere qual è il tuo formato di default. Puoi scegliere fra diversi formati usando `Output_File_Type=x` o `+Fx` dove x è una delle seguenti opzioni...

`+FC`	Targa a 24 bit compresso (RLE)
`+FN`	PNG (portable network graphics)
`+FP`	formato PPM Unix
`+FS`	formato specifico del sistema come Pict Macintosh o BMP di Windows

+FT	Targa a 24 bit non compresso

Nota che i vecchi formati "dump" (+FD) e "raw" (+FR) sono stati abbandonati perché erano usati raramente e non sono più necessari. Sono stati aggiunti i formati PPM, PNG e specifici del sistema. Le immagini in formato PPM non sono compresse e hanno una semplice intestazione in formato testo che le rende un formato ampiamente portabile. PNG è un nuovo formato che è stato progettato non solo per sostituire il formato GIF, ma anche per correggere i suoi difetti. Il formato PNG offre la più alta compressione disponibile senza perdita di qualità per applicazioni come il ray-tracing. Il formato specifico del sistema dipende dalla piattaforma usata e viene trattato nella documentazione specifica.

La maggior parte di questi formati ha un'output a 24 bit per punto, con 8 bit assegnati per i dati di rosso, ed altrettanti per verde e blu. Il formato PNG permette comunque di specificare da 5 a 16 bit per i valori di rosso, verde e blu, dando colori da 15 a 48 bit per punto. La profondità del colore è per default di 8 bit per colore (24 bit per punto, 16 milioni di colori), ma può essere cambiata nel formato PNG impostando `Bits_Per_Color=n` o specificando `+FNn`, dove n è la profndità di colore desiderata.
Specificare una minore profondità di colore come 5 bit/colore (32768 colori) può essere abbastanza per chi ha possibilità di visualizzare a schermo immagini a 8 o 16 bit (256 o 65536 colori) e migliorerà la compressione del file PNG. Maggiori profondità di colore come 10 o 12 bit/pixel possono essere migliori per applicazioni professionali e 16 bit/colore è ottimale per l'output a toni di grigio (vedi il paragrafo "Campi di Quota" per dettagli).
Il formato Targa permette anche l'output di 8 bit/pixel di dati per il canale alfa (trasparenza), mentre il formato PNG ne permette 5 o 16, in dipendenza dalla profondità di colore, come è stato specificato prima. Si può attivare questa opzione con `Output_Alpha=on` oppure `+UA`. Il valore predefinito è `off` o `-UA`. Vedi il paragrafo "Usare il Canale Alfa" per maggiori dettagli sulla trasparenza.
I file PNG memorizzano anche il valore di gamma del sistema oltre alla profondità di colore variabile, al supporto dei formati in toni di grigio, al canale alfa, cosicché l'immagine possa essere visualizzata correttamente su tutti i sistemi (vedi il paragrafo "Impostazioni Hardware").Anche l'impostazione `hf_gray_16`, descritta nel paragrafo "Hf_Gray_16" influenzerà il tipo di dati scritti nel file di output.

6.2.2.3.2 Nome del File di Output

Output_File_Name=file	Imposta il nome del file di output a "file"
+Ofile	idem

Il nome predefinito del file di output è dedotto dal nome della scena e non è necessario specificarlo. Il nome della scena è il nome del file di input insieme alla specificazione di drive, percorso ed estensione. Per esmpio, se il nome del file di input è **c:\povray3\mystuff\myfile.pov**, il nome della scena sarà **myfile**. L'appropriata estensione sarà aggiunta al nome della scena basandosi sul tipo di file. Per esempio, potrebbe essere **myfile.tga** o **myfile.png**.
Si può cambiare il nome di default del file di output usando l'opzione `Output_File_Name=file` oppure `+Ofile`. Per esempio :

```
Input_File_Name=myinput.pov
Output_File_Name=myoutput.tga
```

Se viene specificato un nome di file di output "-" (un segno meno), allora l'immagine verrà scritta all'output standard, solitamente lo schermo. L'output può quindi essere diretto in un altro programma o all'interfaccia grafica, se desiderato.

6.2.2.3.3 Buffer del File di Output

Buffer_Output=bool	Attiva/disattiva il buffer di output (on/off)
+B	Attiva il buffer di output
-B	Disattiva il buffer di output
Buffer_Size=n	Imposta la dimensione del buffer a 'n' kilobyte. Se n è zero, non crea il buffer. Se n è minore di quanto definito dal sistema, usa il default di sistema
+Bn	Attiva il buffer, ne imposta la dimensione ad n
-Bn	Disattiva il buffer, (nelle versioni future, ne imposta la dimensione ad n)

Le opzioni `Buffer_Output` e `Buffer_Size` e il parametro `+B` permettono di assegnare grandi buffer di memoria al file di output. Questo riduce il tempo speso nello scrivere sul disco. Se questo parametro non è specificato, allora quando ogni riga è finita, la linea viene scritta sul file ed il buffer viene vuotato. Sulla maggior parte dei sistemi, questa operazione assicura che il file venga scritto su disco cosicché in caso di un crash del sistema o di altri eventi catastrofici, almeno una parte dell'immagine viene salvata sul disco e quindi è possibile recuperarla. L'impostazione predefinita è di *non* usare il buffer.

6.2.2.4 Istogramma di Impiego della CPU

L'istogramma di utilizzazione della CPU è un modo di capire dove POV-Ray impiega il tempo di rendering e anche un modo interessante per fare height fields. L'istogramma divide lo schermo in una griglia rettangolare. Mentre POV-Ray renderizza l'immagine l'istogramma calcola la quantità di tempo impiegato per renderizzare ciascun pixel ed aggiunge questo punto al tempo totale di rendering per ogni blocco della griglia. Quando il rendering è completo, l'istogramma è un file che rappresenta quanto tempo è stato impiegato per calcolare i pixel in ogni blocco della griglia. Non tutte le versioni di POV-Ray permettono la creazione di istogrammi. Il formato del file di output per l'istogramma dipende dal tipo di file e dal sistema su cui POV-Ray viene eseguito.

6.2.2.4.1 Tipo di File

`Histogram_Type=x`	Imposta il formato del file dell'istogramma, se il formato è "x" non viene creato il file di output dell'istogramma
+HTx	Lo stesso di `Histogram_Type=x`

Il formato del file di output dell'istogramma è quasi sempre lo stesso di quello usato per l'output dell'immagine (vedi "Tipo di File di Output"). I formati disponibili sono :

`+HTC`	CSV valori numerici separati da virgole, usato spesso nei fogli di calcolo
`+HTN`	PNG a toni di grigio
`+HTP`	PPM di Unix
`+HTS`	Specifici del sistema come Pict per Macintosh o BMP per Windows
`+HTT`	Targa-24 bit non compresso (TGA)
`+HTX`	Non viene creato un file di output dell'istogramma.

Il parametro `+HTC` non genera un file in formato Targa non compresso, ma un file di testo formato da una lista separata da virgole che specifica il tempo impiegato per ogni blocco della griglia, ordinata da sinistra a destra e dall'alto verso il basso. Le unità di tempo del file di output in formato

CSV sono dipendenti dal sistema ; è necessario vedere dunque per maggiori dettagli le specifiche documentazioni del sistema sulle unità di tempo utilizzate dai file CSV.
I formati Targa e PPM sono gli stessi utilizzati da POV-Ray per gli height field (Vedi "Height Field"), quindi le informazioni del file dell'istogramma sono convertite in colori ed è impossibile la lettura. Quando sono usate per gli height field, i valori più bassi corrispondono al minore tempo utilizzato per il calcolo dei pixel in quel blocco e valori più alti indicano un maggior tempo impiegato nel blocco corrispondente.
Le immagini in formato PNG sono invece immagini in toni di grigio e sono comode sia per la lettura dei dati sia per la creazione di height field. Nei file PNG le aree più scure (o più basse) indicano una minore quantità di tempo impiegata per quel blocco e le aree più chiare (o più alte) indicano un maggior tempo impiegato nel blocco corrispondente.

6.2.2.4.2 Nome del File

`Histogram_Name=file`	Imposta il nome del file contenente l'istogramma a file
`+HNfile`	Lo stesso di `Histogram_Name=file`

Il nome del file dell'istogramma è il nome del file in cui vengono scritti i dati riguardanti l'istogramma. Se non viene specificato alcun nome verrà utilizzato quello di default che è **histgram.ext**, dove **ext** dipende dal tipo di file specificato con i parametri contenuti nel paragrafo precedente. E' da notare che se il nome del file è specificato, l'estensione del file dovrà essere uguale a quella specificata nel parametro del formato del file.

6.2.2.4.3 Dimensioni della Griglia

`Histogram_Grid_Size=xx.yy`	Imposta le dimensioni della griglia da xx a yy
`+HSxx.yy`	Lo stesso che `Histogram_Grid_Size=xx.yy`

La dimensione della griglia determina la quantità di porzioni in cui l'immagine verrà suddivisa sia in verticale che in orizzontale. Per esempio

```
povray +Isample +W640 +H480 +HTN +HS160.120 +HNhistogrm.png
```

dividerà l'immagine in 160x120 blocchi, ognuno dei quali sarà 4x4 pixel, il file di output sarà in formato PNG, utilizzabile sia per la lettura diretta dei dati che per la creazione di height field. Assegnando numeri più piccoli alla dimensione della griglia si avranno blocchi con un maggior numero di pixel. Utilizzando come file di output un file in formato CSV, il numero di valori contenuti nel file di output sarà lo stesso del numero di blocchi della griglia. Per gli altri formati la dimensione dell'immagine dell'istogramma sarà la stessa di quella dell'immagine renderizzata, per facilitare il confronto tra l'immagine dell'istogramma e l'immagine renderizzata. Se la dimensione della griglia dell'istogramma non è specificata, sarà considerato come se ci fosse un blocco della griglia per pixel.
In sistemi multi-tasking o task-switching l'istogramma potrebbe non rappresentare correttamente l'esatto tempo impiegato da POV-Ray in un dato blocco dal momento che l'istogramma è basato sul tempo reale e non su quello di CPU. Quindi il tempo potrebbe essere stato impiegato anche per altri programmi gestiti dal sistema operativo che sono in esecuzione nello stesso momento. Questo causerà macchie e disturbi di colore nel file dell'istogramma. Questo effetto può essere ridotto riducendo la dimensione della griglia, in modo da avere molti pixel in un blocco.

6.2.3 Opzioni per l'Analisi della Scena

POV-Ray legge il file della scena e si crea un modello interno della scena stessa. Questo processo di analisi è chiamato **parsing**. Mentre avviene il parsing di un file è possibile leggerne altri. Questa

sezione si occupa di chiarire su quali elementi viene fatto il parsing, dove trovarli e quali assunzioni specifiche bisogna fare mentre avviene il parsing.

6.2.3.1 Nome del File di Input

`Input_File_Name=file`	Imposta il nome del file di input a file
`+Ifile`	Lo stesso di `Input_File_Name=file`

probabilmente imposterai sempre questa opzione, ma se non lo fai il nome del file di input predefinito è **object.pov**. Nei sistemi operativi sensibili alla differenza maiuscolo-minuscolo sono definiti sia .POV che .pov. e' necessario specificare *l'intero percorso del file* (per esempio sui sistemi MS-DOS l'impostazione sarebbe **+Ic:\povray3\mystuff\myfile.pov**). specificare il nome del file di input significa anche specificare il nome del file della scena.
Il nome del file della scena è uguale al nome del file di input ma in esso si omettono percorso e formato. Nell'esempio precedente il nome del file della scena è **myfile**. Questo nome è usato per creare un nome predefinito del file di output.
Se viene usato "-" come nome del file di input, si userà l'input standard. In questo modo si può redirigere una scena creata da un altro programma a POV-Ray e renderizzarla senza creare un file di input. Sotto MS-DOS si può provare questa funzione scrivendo

```
type UNASCENA.POV | povray +I-
```

6.2.3.2 Percorsi delle Librerie

`Library_Path=path`	aggiunge un nuovo percorso alla lista dei percorsi
`+Lpath`	Lo stesso di `Library_Path=path`

POV-Ray cerca i file nella directory attiva. Se non li trova legge nella lista dei percorsi ogni altra directory che puoi specificare. POV-Ray non cerca i percorsi definiti dal sistema operativo. Cerca solo nella directory attiva e in quelle che vengono specificate con questa opzione. Per esempio i file include standard sono contenuti in una specifica directory. Puoi dire a POV-Ray di cercarli là con...

```
Library_Path=c:\povray3\include
```

non si deve specificare nessuna barra (\ o /) alla fine del percorso.
La specificazione di molti percorsi non sovrascrive i precedenti che rimangono tutti validi. Possono essere specificati fino a dieci percorsi. Se si specifica lo stesso percorso due volte, questo conterà per uno. POV-Ray cercherà per primo nella directory attiva e in seguito in quelle specificate, nell'ordine in cui sono state specificate.

6.2.3.3 Versione del Linguaggio

`Version=n.n`	permette la piena compatibilità con versioni precedenti di POV-Ray specificate con n.n
`+MVn.n`	Lo stesso di `Version=n.n`

Nonostante che nella versione di POV-Ray 3.0 siano stati fatti molti cambiamenti nel linguaggio e nella sintassi, resta ancora possibile utilizzare la sintassi delle precedenti versioni 2.0 e 1.0. Quando è possibile si è cercato di mantenere la piena compatibilità con le precedenti versioni. Una delle funzioni della versione 2.0 che è incompatibile con qualunque file di una scena creato con la versione 1.0 è il parsing di espressioni a virgola mobile. Impostando `Version=1.0` o usando il parametro `+MV1.0` si potranno evitare le funzioni del parsing non presenti nella versioni 1.0 e molti dei messaggi di attenzione e così quasi tutti i file della versione 1.0 potranno funzionare. I

cambiamenti fra la versione 2.0 e 3.0 sono meno fondamentali. Impostare `Version=2.0` è necessario solo per eliminare qualche messaggio di attenzione. Naturalmente l'impostazione predefinita per questa opzione è `Version=3.0`.
Questa opzione influenzerà le impostazioni iniziali mentre la funzione `#version` renderà possibile cambiare anche più volte all'interno della scena il tipo di linguaggio e di sintassi compatibile. Vedere la sezione "Istruzioni di Versione".

6.2.3.4 Rimuovere il Bounding Impostato Manualmente

`Remove_Bounds=bool`	Permette di rimuovere i bounds non necessari (on/off)
`+UR`	Permette di rimuovere i bounds non necessari
`-UR`	Permette di non rimuovere i bounds non necessari
`Split_Unions=bool`	permette l'unione di bounds separate (on/off)
`+SU`	permette l'unione di bounds separate
`-SU`	permette l'unione di bounds separate

Le precedenti versioni di POV-Ray non avevano un sistema automatico per suddividere lo spazio in regioni e aumentare la velocità delle prove di intersezione tra i raggi e gli oggetti. Coloro che utilizzavano le precedenti versioni di POV-Ray dovevano, per rendere il rendering più veloce, creare tali suddivisioni manualmente. POV-Ray 3.0 ha capacità molto più sofisticate rispetto alle precedenti versioni. In molti casi la suddivisione automatica è molto più veloce della suddivisione manuale delle precedenti versioni. Comunque POV-Ray rimuove la suddivisione automatica quando non è necessaria. In casi rari è possibile che si preferisca mantenere la suddivisione manuale. Alcune scene più vecchie usano scorrettamente la suddivisione. Se POV-Ray rimuove la suddivisione della scena l'immagine non sarà più la stessa. Per disattivare la suddivisione automatica è necessario specificare `Remove_Bounds=off` o usare il parametro `-UR`. Il valore predefinito è `Remove_Bounds=on`.

Un argomento su cui c'è ancora indecisione è la suddivisione delle unioni CSG cui è stato applicato manualmente il bounding. Le unioni non suddivise sono sempre divise nelle parti che le compongono in modo che la suddivisione automatica lavori meglio. Molti non dividono le unioni perché sanno che così facendo verrà rallentato il tempo di esecuzione del file. Se un'unione viene manualmente suddivisa si suppone che veramente si voglia fare questa operazione, quindi non viene rimossa tale divisione. Se vuoi togliere la suddivisione manuale dalle unioni devi specificare `Split_Unions=on` o usare `+SU`. Il valore di default è `Split_Unions=off`.

6.2.4 Uscita al Sistema Operativo

`Pre_Scene_Command=s`	imposta il commando prima dell'intera scena
`Pre_Frame_Command=s`	imposta il comando prima di ogni fotogramma
`Post_Scene_Command=s`	Imposta i comandi dopo l'intera scena
`Post_Frame_Command=s`	imposta il comando dopo ogni fotogramma
`User_Abort_Command=s`	imposta il comando quando viene interrotto POV-Ray
`Fatal_Error_Command=s`	imposta il comando quando POV-Ray ha un errore fatale.

Per questa opzione non sono disponibili i parametri (+/-) e non possono essere usati dalla linea di comando. Possono solo essere usati da file **.ini**.
POV-Ray dà la possibilità di uscire al sistema operativo in determinati punti chiave per eseguire un

altro programma o un file batch. Solitamente questo è usato per gestire file creati dal loop interno delle animazioni e comunque questi comandi sono utilizzabili per ogni scena. Il comando è una singola linea di testo che viene passata al sistema operativo per eseguire un programma. Per esempio

```
Post_Scene_Command=tga2gif -d -m myfile
```

userà l'utilità **tga2gif** con `-D` e `-M` per convertire **myfile.tga** a **myfile.gif** dopo che è stato completato il rendering della scena.

6.2.4.1 Sostituzione di Stringhe nei Comandi di Shell

Può essere noioso cambiare l'opzione `Post_Scene_Command` ogni volta che viene cambiato il nome del file di scena. POV-Ray può sostituire automaticamente alcuni valori nelle righe di comando. Per esempio :

```
Post_Scene_Command=tga2gif -d -m %s
```

POV-Ray sostituirà il `%s` con il nome del file di scena. Tale nome è specificato nell' opzione INI `Input_File_Name` o nel parametro `+I` e verrà utilizzato dal `Post_Scene_Command` eliminando dal nome il percorso. Per esempio :

```
Input_File_Name=c:\povray3\scenes\waycool.pov
```

diventerà...

```
Post_Scene_Command=tga2gif -d -m waycool
```

Per le animazioni può essere necessario avere il nome esatto del file di output con l'appropriato numero di fotogramma incluso. Supponendo che si desideri salvare le immagini di output in un file .zip usando **pkzip**, si potrebbe fare...

```
Post_Frame_Command=pkzip -m %s %o
```

dopo aver renderizzato il fotogramma 12 di **myscene.pov**, POV-Ray passerà al sistema operativo con "**pkzip -m myscene mysce012.tga**". Il parametro `-M` in **pkzip** sposta **mysce012.tga** a **myscene.zip** e lo cancella dalla directory. Da notare che `%o` aggiunge i numeri al nome del fotogramma solo quando si tratta di un'animazione. Per le opzioni `Pre_Scene_Command` e `Post_Scene_Command` non c'è nessun numero di fotogramma, così verrà usato il nome dell'originale file di output senza aggiunta di un numero. Ogni comando di interruzione o di errore fatale (`User_Abort_Command` o `Fatal_Error_Command`) non nel loop di animazione darà una sostituzione del nome senza numerazione.
Questa è la lista completa delle possibili sostituzioni.

`%o`	Il file di output avrà estensione e numero di fotogramma se siamo in un'animazione
`%s`	il nome dell'output deriverà dal nome del file di input
`%n`	numerazione (per animazione) che cresce con i fotogrammi
`%k`	valore di clock per questo file di output
`%h`	altezza dell'immagine in pixel

%w	larghezza dell'immagine in pixel
%%	un solo segno di %.

6.2.4.2 Comandi di Shell in Sequenza

Questa è la sequenza di azioni in un loop di animazione. Le scene non animate funzionano allo stesso modo se non che non c'è il loop proprio dell'animazione.

1) Vengono esaminati tutti i file **.ini**, le opzioni ed i parametri una volta.
2) Viene aperto ogni output di testo e si crea il file **.ini** se necessario.
3) Viene eseguito Pre_Scene_Command se c'è.
4) Loop dell'animazione (o solo una ripetizione, se la scena non è animata).
a) Viene eseguito Pre_Frame_Command se c'è.
b) Si esegue il parsing dell'intero file della scena, si apre il file di output e si leggono le impostazioni, si attivano le impostazioni di schermo per la visualizzazione, viene renderizzata l'immagine, si distruggono tutti gli oggetti, le texture ecc., vengono chiusi il file di output e il display.
c) Si esegue il `Post_Frame_Command` se c'è.
d) Si ripete dal punto 4 finchè non sono stati rederizzati tutti i fotogrammi.
5) Si esegue il Post_Scene_Command se c'è.
6) Uscita da POV-Ray.

Se è presente il comando `User_Abort_Command,` il processo può essere interrotto quando lo si ritenga necessario. Le interruzioni possono essere eseguite dolo durante il parsing e nelle parti del rendering che seguono il passaggio 4.
Se si incorre in un errore fatale ed è presente il comando `Fatal_Error_Command` allora questo è eseguito. In qualche caso un errore nella memoria o altro può essere la causa di un crash totale del programma, in questo caso non c'è nessuna possibilità di uscita. Gli errori fatali possono essere annidati in ogni punto del processo inclusi i comandi, le opzioni e i parametri.
E' da notare che viene fatto il parsing dell'intera scena per ogni fotogramma. Le versioni future di POV-Ray permetteranno di memorizzare parti di un fotogramma per passarle al successivo, ma per ora ricomincia tutto il processo ogni volta. Anche il comando `Pre_Frame_Command` viene eseguito prima del parsing. E' possibile usarlo per richiamare alcune utilità per la generazione della scena del fotogramma corrente. Queste utilità possono sovrascrivere via via il tuo file .pov o .inc, se necessario. Ad esempio, potresti volere una nuova immagine .gif o .tga da ogni fotogramma da utilizzare per mappe di immagini o per gli height field.

6.2.4.3 Azioni di Ritorno dei Comandi di Shell

Pre_Scene_Return=s	Imposta i comandi che POV-Ray esegue in caso di errore durante la fase antecedente alla scena
Pre_Frame_Return=s	Imposta i comandi che POV-Ray esegue in caso di errore durante la fase antecedente ad un fotogramma
Post_Scene_Return=s	Imposta i comandi che POV-Ray esegue in caso di errore durante la fase seguente alla scena
Post_Frame_Return=s	Imposta i comandi che POV-Ray esegue in caso di errore durante la fase seguente ad un fotogramma
User_Abort_Return=s	Imposta i comandi che POV-Ray esegue in caso di interruzione della scena da parte di colui che utilizza il programma
Fatal_Error_Return=s	Imposta i comandi che POV-Ray esegue in caso di errore fatale

Per questi comandi non esistono i corrispondenti parametri. Non possono quindi essere usati nella linea di comando, ma solamente impostati nei file **.ini**.
Molti sistemi operativi specificano in caso di errore un codice che meglio definisca il tipo di errore in cui si è incorsi. Quando POV-Ray esegue un comando di una shell può far uso di questi codici e regolarsi di conseguenza a seconda che il codice sia zero o no. Anche POV-Ray utilizza questi codici e considera un codice di errore di zero come operazione eseguita con successo, 1 per un errore fatale e 2 per l'interruzione causata dall'utente.
Queste azioni sono impostate scrivendo una sola lettera nell'opzione relativa (una di quelle elencate qui sopra). Le azioni possibili sono :

`I`	ignorare il codice
`S`	evitare un passo
`A`	evitare tutti i passi
`Q`	chiudere POV-Ray immediatamente
`U`	l'utente ha interrotto il programma
`F`	si è creato un errore fatale in POV-Ray

Per esempio se l'impostazione `Pre_Frame_Command ri`chiama un programma esterno che genera gli height-field e questa utility fallisce, allora si avrà un codice diverso da zero. E' possibile che a questo punto si desideri comunque interrompere il processo. L'opzione `Pre_Frame_Return=F` permetterà a POV-Ray di interrompere l'esecuzione dell'operazione nel caso in cui l'esecuzione di `Pre_Frame_Command` dia luogo ad un codice diverso da zero.
Qualche volta si può gioire per un codice diverso da zero. Supponendo ad esempio che si voglia vedere se un determinato fotogramma sia già stato renderizzato, allora si può usare la lettera `S` per evitare questo fotogramma nel caso sia già stato renderizzato. Molti programmi di utilità riportano errori se non si è riusciti a trovare un determinato file. Per esempio il comando

```
pkzip -V myscene mysce012.tga
```

richiama **pkzip** per vedere l'archivio **myscene.zip** per il file **mysce012.tga**. Se il file non è nell'archivio **pkzip** darà un codice diverso da zero.
Altre volte è possibile voler interrompere il ciclo se il file è stato trovato. Quindi sarà necessario invertire l'azione in modo che eviti di continuare se il codice è zero e continui se non è zero. Per compiere questa inversione è necessario premettere alla lettera il segno "-" (anche "!" determina la stessa azione dal momento che in molti programmi è proprio questo il segno di negazione).

`Pre_Frame_Return=S`	POV-Ray si fermerà se il codice segnala un errore (non-zero) e procederà normalmente se non ci sono errori (zero).
`Pre_Frame_Return=-S`	POV-Ray si fermerà se il codice non segnala errori (zero) e procederà normalmente se ci sono errori (non-zero).

La lettera assegnata di default a tutte le opzioni è `I`, che significa che qualunque errore si sia verificato durante l'utilizzo di un programma di utilità questo *verrà ignorato*. POV-Ray continuerà semplicemente a fare ciò che stava facendo prima dell'utilizzo di quel programma, le altre azioni dipenderanno dal contesto. E' anche possibile che si desideri riferirsi al loop dell'animazione nella sezione precedente. L'azione per ogni shell è la seguente.

Al ritorno di `User_Abort_Command` se c'è un'azione che è stata specificata...

`F` trasforma l'interruzione dell'utente in un errore fatale.

Esegue il comando `Fatal_Error_Command`, se c'è.
Esce da POV-Ray con codice 1.

`S`, `A`, `Q`, o `U` procede con l'interruzione. Esce da POV-Ray con codice 2.

Di ritorno dal `Fatal_Error_Command` procede comunque con l'errore fatale. Esce da POV-Ray con codice 1.

Di ritorno dal `Pre_Scene_Command`, `Pre_Frame_Command`, `Post_Frame_Command` o `Post_Scene_Commands` se c'è un'azione che è stata specificata...

`F` genera un errore fatale.
Esegue il comando `Fatal_Error_Command`, se c'è.
Esce da POV-Ray con codice 1.

`U` crea un'interruzione dell'utente.
Esegue il comando `User_Abort_Command`, se c'è.
Esce da POV-Ray con un codice 2.

`Q` chiude POV-Ray immediatamente.
Si comporta come se POV-Ray non fosse mai stato eseguito.
Non esegue i successivi comandi di shell (nemmeno `Post_Scene_Command`) ed esce da POV-Ray con errore di codice 0.

Di ritorno dal comando `Pre_Scene_Command` se è stata specificata un'azione allora...

`S` evita di renderizzare tutti i fotogrammi.
Si comporta come se tutti i fotogrammi fossero renderizzati normalmente.
Non esegue `Pre_Frame_Command` o `Post_Frame_Commands`.
Eseguirà il comando `Post_Scene_Command`, se c'è.
Esce da POV-Ray con codice 0.
Nelle precedenti tabelle significava tralasciare il passo #4.

`A` tralascia tutta la scena. E' identico a `Q` (quit).
Nelle precedenti tabelle significava evitare tutti i passi fino al #6.

Di ritorno dal comando `Pre_Frame_Command` se è stata specificata un'azione allora...

`S` tralascia solo questo fotogramma.
Facendo così questo fotogramma non verrà creato.
Non esegue il comando `Post_Frame_Command`.
Procede con il fotogramma successivo.
Nelle precedenti tabelle significa saltare il passo #4b e #4c ma tornare indietro come necessario per l'animazione al passo #4d.

`A` non renderizza questo fotogramma e tutti gli altri.
In questo modo la scena viene completata normalmente.
Non viene eseguito nessun comando impostato con `Post_Frame_Commands`.
Viene eseguito il `Post_Scene_Command`, se c'è.
Si esce da POV-Ray con un codice di errore 0.

Nelle precedenti tabelle questo significa evitare il resto del passo #4 e procedere al passo #5.

Di ritorno dal comando `Post_Frame_Command` se è stata specificata un'azione allora...

`S` tralascia tutti i fotogrammi seguenti. Facendo così l'animazione verrà creata normalmente.
Esegue il comando `Post_Scene_Command`.
Esce da POV-Ray con codice 0.
Nella tabella precedente questo significa tralasciare dal passo #4 e procedere al passo 5.

`A` lo stesso che `S`.

Di ritorno dal comando `Post_Scene_Command` se c'è un'azione impostata allora..

`S` o `A` lo stesso di `I`.

6.2.5 Output di Testo

L'output è il modo in cui POV-Ray ti informa riguardo a cosa sta facendo, cosa farà e cosa ha fatto. Questa versione di POV-Ray divide i messaggi in sette parti separate. Alcune versioni distinguono queste parti con colori diversi, alcune permettono di sfogliare le varie pagine di messaggi. Tutte le versioni ti permettono di attivare o disattivare alcuni di questi messaggi o di poter creare una copia di questi messaggi in un file o in più di uno. Questa sezione descriverà le opzioni che ti permetteranno di controllare l'output dei messaggi.

6.2.5.1 Tipi di Output di Testo

Ci sono sette parti distinte che possono essere evidenziate nei messaggi di output di POV-Ray. In alcun versioni queste sono evidenziate da colori diversi. Il testo è mostrato quando è necessario, quindi ci possono essere vari tipi di messaggi mescolati. La distinzione è importante solo se si decide di disattivare alcune opzioni. Su alcuni sistemi operativi si possono rivedere in un buffer separato.
Qui c'è una descrizione di ogni tipo :

Banner : questa parte dei messaggi mostra la schermata iniziale (banner) del programma, il copyright, la lista di coloro che hanno contribuito alla realizzazione e alcuni messaggi di aiuto. Non può essere disattivata o inserita in un file di output perché molto del testo che vi compare viene mostrato ancora prima che POV-Ray legga le opzioni o i parametri, così come non è possibile avere un'opzione od un parametro che la controlli. Ci sono, però parametri che mostrano la parte relativa agli aiuti. Sono trattati nella sezione "Comandi per le Schermate di Aiuto".

Debug : questa parte mostra i messaggi relativi alla correzione del programma (debug). E' stata ideata principalmente per coloro che sviluppano il programma, ma può essere usata per mostrare messaggi durante l'analisi del file della scena. Vedere la sezione "Messaggi di Testo" per ulteriori informazioni. Questa parte può essere attivata o disattivata o trascritta in un file di testo.

Fatal : mostra gli errori fatali. Dopo aver mostrato questo testo POV-Ray si fermerà. Quando l'errore si è verificato durante il parsing di una scena, verranno mostrate le righe della scena che contengono l'errore. Anche questa parte può essere attivata o disattivata o salvata in un file di testo.

Render : mostra le informazioni riguardanti le opzioni che sono state specificate per il rendering. Include richiami a tutte le più importanti opzioni tra cui il nome della scena, la risoluzione, le impostazioni delle animazioni, l'anti-aliasing e altre. Anche questa parte può essere attivata o disattivata o salvata in un file di testo.

Statistiche : mostra le statistiche dopo che un fotogramma è stato renderizzato. Include informazioni sul numero di raggi tracciati, la quantità di tempo impiegata ed altre. Anche questa parte può essere attivata o disattivata o salvata in un file di testo.

Status : mostra una linea di stato che spiega che cosa stia facendo il programma in quel momento, sul alcuni sistemi questa linea è mostrata in una barra di stato che è situata in fondo allo schermo. Questa non può essere copiata in un file di testo, anche perché generalmente non ce n'è alcun bisogno. Il testo è mostrato in virtù dell'opzione `Verbose` o del parametro `+V`, quindi può essere disattivata.

Warning : questa parte mostra messaggi di attenzione relativi al parsing di una scena o ad altro. Nonostante questi messaggi POV-Ray può continuare il rendering. Sarai informato se POV-Ray estrapolerà determinate assunzioni riguardo alla tua scena in modo da poter continuare. In generale ogni volta che si vede un messaggio di attenzione si può prevedere che future versioni di POV-Ray non permetteranno lo stesso errore. Anche questa parte può essere attivata o disattivata o mostrata in un file di testo.

6.2.5.2 Output di Testo su Console

`Debug_Console=bool`	rende o non rende possibile la visualizzazione dei messaggi di debug (on/off)
`+GD`	lo stesso di `Debug_Console=On`
`-GD`	lo stesso di `Debug_Console=Off`
`Fatal_Console=bool`	rende o non rende possibile la visualizzazione dei messaggi di errore fatale (on/off)
`+GF`	lo stesso di `Fatal_Console=On`
`-GF`	lo stesso di `Fatal_Console=Off`
`Render_Console=bool`	rende o non rende possibile la visualizzazione dei messaggi di rendering (on/off)
`+GR`	lo stesso di `Render_Console=On`
`-GR`	lo stesso di `Render_Console=Off`
`Statistic_Console=bool`	rende o non rende possibile la visualizzazione dei messaggi di delle statistiche (on/off)
`+GS`	lo stesso di `Statistic_Console=On`
`-GS`	lo stesso di `Statistic_Console=Off`
`Warning_Console=bool`	rende o non rende possibile la visualizzazione dei messaggi di attenzione (on/off)
`+GW`	lo stesso di `Warning_Console=On`
`-GW`	lo stesso di `Warning_Console=Off`
`All_Console=bool`	rende o non rende possibile la visualizzazione dei messaggi di debug, errori fatali, rendering, statistiche e messaggi di attenzione (on/off).
`+GA`	lo stesso di `All_Console=On`
`-GA`	lo stesso di `All_Console=Off`

E' possibile eliminare la visualizzazione dei messaggi relativi al debug, agli errori fatali, al rendering, alle statistiche o ai messaggi di attenzione (warning). Per esempio l'opzione

Statistic_Console=off o il parametro -GS possono disabilitare i messaggi statistici. Usando on o +GS si possono attivare di nuovo. si può allo stesso modo attivare o disattivare tutti gli altri cinque tipi di messaggi utilizzando l'opzione All_Console o il parametro +GA. Queste opzioni hanno effetto immediato una volta specificate. Ovviamente un messaggio di errore o di attenzione che si è verificato prima del cambiamento dell'impostazione non ne risulterà variato.

6.2.5.3 Dirigere l'Output di Testo su File

Debug_File=true	invia il testo relativo alle informazioni di debug al file debug.out
Debug_File=false	disattiva il testo relativo ai messaggi di debug
Debug_File=file	invia il testo relativo alle informazioni di debug a file
+GDfile	sia Debug_Console=On, sia Debug_File=file
-GDfile	sia Debug_Console=Off, sia Debug_File=file
Fatal_File=true	invia il testo relativo alle informazioni di errori fatali al file fatal.out
Fatal_File=false	disattiva il testo relativo ai messaggi di errori fatali
Fatal_File=file	invia il testo relativo alle informazioni di errori fatali a file
+GFfile	sia Fatal_Console=On, sia Fatal_File=file
-GFfile	sia Fatal_Console=Off, sia Fatal_File=file
Render_File=true	invia il testo relativo alle informazioni sul rendering al file render.out
Render_File=false	disattiva il testo relativo al rendering
Render_File=file	invia il testo relativo alle informazioni sul rendering a file
+GRfile	sia Render_Console=On, sia Render_File=file
-GRfile	sia Render_Console=Off, sia Render_File=file
Statistic_File=true	invia il testo relativo alle informazioni sulle statistiche al file stats.out
Statistic_File=false	disattiva il testo relativo alle statistiche
Statistic_File=file	invia il testo relativo alle informazioni sulle statistiche a file
+GSFile	sia Statistic_Console=On, sia Statistic_File=file
-GSFile	sia Statistic_Console=Off, sia Statistic_File=file
Warning_File=true	invia il testo relativo alle informazioni sui messaggi di attenzione al file warning.out
Warning_File=false	disattiva il testo relativo ai messaggi di attenzione
Warning_File=file	invia il testo relativo alle informazioni sui messaggi di attenzione a file

+GWfile	sia Warning_Console=On, sia Warning_File=file
-GWfile	sia Warning_Console=Off, sia Warning_File=file
All_File=true	invia il testo relativo alle informazioni sui messaggi di attenzione, debug, errori fatali, rendering, statistiche al file **alltext.out**
All_File=false	disattiva il testo relativo ai messaggi di attenzione, debug, errori fatali, rendering, statistiche al file
All_File=file	invia il testo relativo alle informazioni sui messaggi di attenzione, debug, errori fatali, rendering, statistiche a **file**
+GAfile	sia All_Console=On, sia All_File=file
-GAfile	sia All_Console=Off, sia All_File=file

Si può inviare una copia dei messaggi a un file per i messaggi di debug, errore fatale, rendering, statistiche o di attenzione. Per esempio utilizzando l'opzione Statistic_File=s o il parametro +GSs. Se la stringa s è true o uno degli altri parametri validi allora si potranno inviare i messaggi ad un file con nome predefinito. Valori validi sono true, yes, on o 1. Se il valore è false non si avrà file di output. Valori validi per false are false, no, off o 0. Ogni altra stringa specificata attiva il file di output e la stringa è interpretata come il nome del file di output.
Allo stesso modo si possono specificare true, false od il nome del file dopo un parametro, per esempio +GSfile. E' anche possibile mandare ad uno stesso file tutti e cinque i tipi di messaggio con l'opzione All_File o il parametro +GA. Non devi specificare lo stesso nome di file per più tipi di messaggi perché POV-Ray fallirà nell'aprire o chiudere lo stesso file due volte.
Queste opzioni hanno effetto immediato una volta specificate. Ovviamente un messaggio di errore o di attenzione che si è verificato prima del cambiamento dell'impostazione non ne risulterà variato.

6.2.5.4 Comandi per le Schermate di Aiuto

+H o +?	Mostra la schermata di aiuto 0
da +H0 a +H8	mostra la schermata di aiuto da 0 a 8
da +?0 a +?8	lo stesso di 'da+H0 a +H8'

Non ci sono comandi per i file **.ini** equivalenti a queste opzioni.
I sistemi operativi che hanno un'interfaccia grafica come MacOS e Windows (ma non X-Windows, N.d.T.) hanno una guida in linea. Le altre versioni di POV-Ray hanno solo una piccola e rapida guida a schermo. Il parametro +? Seguito da un numero da 0 a 8, mostrerà questa guida a schermo subito dopo la parte di messaggi relativi alla banner. Dopo aver mostrato questa breve guida POV-Ray si ferma. Alcuni sistemi operativi non permettono che si usi come linea di comando il punto interrogativo, è anche possibile usare +H. questo parametro è anche usato per specificare l'altezza dell'immagine in pixel. Comunque +H sarà interpretato come relativo alla guida se unico parametro presente nella riga e comunque se seguito da un numero compreso tra 0 e 8.

6.2.6 Opzioni di Rendering

Ci sono molti modi di tracciare un raggio. Spesso si deve trovare il giusto compromesso tra qualità e velocità. Spesso opzioni che possono rendere più veloce l'esecuzione diminuiscono la qualità dell'immagine. Questa sezione si occupa delle opzioni che dicono a POV-Ray come tracciare i raggi con l'appropriato rapporto velocità/qualità.v

6.2.6.1 Impostazioni della Qualità

`Quality=n`	imposta il valore della qualità a n (compreso tra 0 ed 11)
`+Qn`	lo stesso di `Quality=n`

Questi comandi permettono di specificare la qualità dell'immagine renderizzata. Puoi scegliere di tenere bassa la qualità dei rendering di prova per poi aumentarla per il rendering finale. Le regolazioni di qualità sono fatte tramite l'eliminazione di alcuni dei calcoli normalmente eseguiti. Per esempio impostare la qualità con un valore inferiore a 4 significherà non renderizzare le ombre. Impostarla sotto a 8 significherà non renderizzare riflessioni e rifrazioni. I valori corrispondono ai seguenti livelli di qualità :

`0, 1`	mostra solo i colori a tinta unita (quick colors), usa solo la luce ambiente. I quick colors sono usati solo fino a 5 o più bassi
`2, 3`	mostra sia la luce ambiente che la luce diffusa
`4`	renderizza le ombre, ma non completamente le luci.
`5`	renderizza le ombre e luci completamente
`6, 7`	calcola i pattern delle texture
`8`	calcola riflessioni, rifrazioni e raggi trasmessi.
`9`	calcola gli aloni.

Il valore predefinito è 9.

6.2.6.2 Impostazioni di Radiosity

`Radiosity=bool`	attiva o disattiva la radiosità (on/off)
`+QR`	attiva la radiosità
`-QR`	disattiva la radiosità

La radiosità è un ulteriore funzione che si occupa di calcolare la riflessione interdiffusa della luce. E' decisamente molto lenta e comunque *ancora sperimentale*. I parametri che controllano come sia calcolata la radiosità sono specificati nella sezione delle impostazioni generali che si occupa di questa particolare funzione. Vedere la sezione "Radiosity" per ulteriori dettagli.

6.2.6.3 Controllo sul Bounding Automatico

`Bounding=bool`	attiva disattiva il bounding (on/off)
`+MB`	attiva il bounding; livello 25 o quanto specificato prima
`-MB`	disattiva il bounding
`Bounding_Threshold=n`	imposta il livello di bounding a n
`+MBn`	attiva il bounding; imposta il livello di bounding a n
`-MBn`	disattiva il bounding; nel futuro imposterà il livello di bounding a n
`Light_Buffer=bool`	attiva o disattiva il buffer delle luci (on/off)
`+UL`	attiva il buffer delle luci

-UL	o disattiva il buffer delle luci
Vista_Buffer=bool	attiva o disattiva il vista buffer (on/off)
+UV	attiva il vista buffer
-UV	disattiva il vista buffer

POV-Ray usa molti sistemi di suddivisione spaziale per aumentare la velocità dei test di intersezione dei raggi con gli oggetti. Il sistema primario usa una gerarchia di scatole annidate le une dentro le altre. Questo sistema racchiude tutti gli oggetti di dimensione finita dentro delle scatole rettangolari invisibili che sono disposte in una gerarchia ad albero (e che sono dette *bounding boxes*). Prima di iniziare a fare le prove con i raggi l'albero (la gerarchia) è discendente e sono calcolati solo quei raggi che colpiscono le scatole. Questo può aumentare molto la velocità del rendering. Però, per scene che contengono comunque pochi oggetti, questo sistema diventa un'inutile complicazione. L'opzione `Bounding=off` o il parametro `-MB` possono disattivarlo. Il valore predefinito è `on`.

L'opzione `Bounding_Threshold=n` o il parametro `+MBn` permettono di impostare il numero minimo di oggetti presenti nella scena superato il quale si usi il bounding, il valore predefinito è 25, che significa che se nella scena ci sono meno di 25 oggetti POV-Ray automaticamente non applicherà il bounding. Generalmente sarebbe buona norma usare una soglia più bassa come ad esempio 5.

Oltre a questo sistema POV-Ray usa altri due metodi di suddivisione spaziale chiamati *light buffering* e *vista buffering*, anch'essi hanno la funzione di rendere più veloce il rendering. Questi sistemi funzionano solo quando è attivato il bounding. Il *vista buffer* è creato proiettando la gerarchia di scatole sullo schermo e determinando la parte di visuale che è coperta da ogni elemento della gerarchia. Solo gli oggetti che sono posti in un dato pixel sono affetti dai raggi primari. Il vista buffer può essere usato solo con telecamere che usano viste prospettiche o ortografiche perché con queste si ha un punto di vista stabile e quindi una proiezione ragionevole (cioè, le linee dritte devono rimanere linee dritte anche dopo la proiezione).

Il *light buffer* è creato racchiudendo ogni fonte di luce in una immaginaria scatola e proiettando la gerarchia delle scatole del bounding in ognuna delle sue sei facce. Questo potrà essere usato solo per punti di luce fissi e non per le area lights.

I raggi riflessi o trasmessi non trarranno alcun vantaggio dall'uso di light o vista buffer.

Le impostazioni predefinite sono `Vista_Buffer=on` o `+UV and Light_Buffer=on` o `+UL`. Le opzioni di disattivazione di questi metodi di calcolo sono a disposizione anche per dimostrare la loro utilità e come protezione per eventuali errori in cui potrebbero incorrere.

In generale ogni oggetto di dimensioni finite e molti degli oggetti creati con operazioni CSG risponderanno senza errori a questo sistema di divisione. Inoltre gli oggetti blob e mesh usano un metodo di bounding interno. Questo non può essere regolato con i comandi che sono stati illustrati sin qui. Possono essere disattivati usando la sintassi appropriata nel file della scena. (vedere "Blob" e "Meshes" per ulteriori dettagli). Anche il testo è diviso nelle singole lettere che sono poi sottoposte alla gerarchia del bounding, anche alcune combinazioni booleane di oggetti finiti e infiniti sono sottoposte al bounding. Il risultato finale è che ti sarà necessario raramente aggiungere manualmente il bounding, come era invece necessario nelle precedenti versioni di POV-Ray, a meno che non usi molti oggetti infiniti.

6.2.6.4 Opzioni sull' Anti-Aliasing

`Antialias=bool`	attiva disattiva l'anti-aliasing (on/off)
`+A`	attiva l'anti aliasing con soglia 0.3 o precedenti
`-A`	disattiva l'anti aliasing
`Sampling_Method=n`	imposta il metodo di campionamento (1 or 2)
`+AMn`	lo stesso di `Sampling_Method=n`
`Antialias_Threshold=n.n`	imposta la soglia di anti-aliasing
`+An.n`	imposta la soglia di anti-aliasing a n.n
`-An.n`	disattiva l'anti-aliasing (soglia di anti-aliasing in futuro)
`Jitter=bool`	attiva o disattiva il jitter (on/off)
`+J`	attiva aa-jitter con 1.0 o quantità precedenti
`-J`	disattiva aa-jitter
`Jitter_Amount=n.n`	imposta la quantità di aa-jitter a n.n. Se n.n <= 0 aa-jitter è disattivato
`+Jn.n`	imposta la quantità di aa-jitter a n.n. Se n.n <= 0 aa-jitter è disattivato
`-Jn.n`	disattiva aa-jitter (stabilirà la quantità di jitter)
`Antialias_Depth=n`	imposta la profondità di anti-aliasing (1 <= n <= 9)
`+Rn`	lo stesso di `Antialias_Depth=n`

Il processo di raytracing è in realtà un processo di campionamento di un'immagine svolto generalmente ad un tasso di campionamento di un campione per pixel. Questo modo di procedere può creare una quantità di errori. Tra questi potrebbe verificarsi che le linee curve non siano smussate ma sembrino a scalini, punti di interferenza tra oggetti, perdita di dettagli o addirittura di oggetti che essendo molto piccoli vengono persi nell'intersezione di due pixel. L'effetto responsabile di questi problemi è l'*aliasing*.

L'anti-aliasing è una tecnica che permette di tentare di risolvere questi problemi o almeno di ridurli. Generalmente l'anti-aliasing permette all'immagine di apparire più smussata. L'opzione `Antialias=on` o il parametro `+A` attivano l'anti-aliasing.

Quando l'anti-aliasing è attivato POV-Ray tenta di ridurre gli errori proiettando molti più raggi in un pixel e quindi facendo una media dei risultati per determinare il colore del pixel. Questa tecnica che è chiamata sovracampionamento (**super-sampling**) può migliorare di molto il risultato delle immagini, ma aumenta il tempo necessario per renderizzare una scena, dal momento che devono essere eseguiti molti più calcoli.

POV-Ray permette di utilizzare due diversi metodi di sovracampionamento. L'opzione `Sampling_Method=n` o il parametro `+Amn` seleziona il *sovracampionamento non adattivo* (metodo 1) o il *sovracampionamento adattivo* (metodo 2). Selezionando uno di questi metodi non si attiva l'anti-aliasing. Lo si attiva invece usando il parametro `+A` o il comando `Antialias=on`.

Per default viene usato il metodo non adattivo (+AM1), POV-Ray traccia inizialmente un raggio per

pixel. Se il colore del pixel è diverso da quelli vicini di più di un determinato valore di soglia allora il pixel è sovracampionato proiettando un numero definito di raggi addizionali. La soglia predefinita è 0.3 ma si può cambiare usando l'opzione `Antialias_Threshold=n.n`. Quando si usano i parametri , il valore di soglia può seguire `+A`. Per esempio, `+A0.1` attiva l'anti-aliasing e imposta il valore di soglia a 0.1.

Le comparazioni riguardanti la soglia sono calcolate come segue. Se r_1, g_1, b_1 e r_2, g_2, b_2 sono le componenti rgb di due pixel allora la differenza tra due pixel è calcolata così :

```
diff = abs(r1-r2) + abs(g1-g2) + abs(b1-b2).
```

Se la differenza è maggiore del valore di soglia prestabilito, allora entrambi i pixel sono sovracampionati. I valori rgb variano tra 0.0 e 1.0, così la più grande differenza possibile tra due pixel è 3.0. Se la soglia dell'anti-aliasing è 0.0 ogni pixel è sovracampionato. Se è 3.0 non ci sarà anti-aliasing. Soglie più basse significano più anti-aliasing e minore velocità di rendering. Usa l'anti-aliasing per la versione finale e non per le prove. Grandi contrasti nella figura possono essere risolti con grandi valori di tolleranza. I valori ottimali potrebbero essere intorno a 0.2 - 0.4.

Quando si usa il metodo non adattivo, il numero predefinito di sovracampionamenti è 9 per pixel, collocati in una griglia 3x3. L'opzione `Antialias_Depth=n` o il parametro `+Rn` controllano il numero di righe e colonne di parti prese per il super-sampling. Per esempio `+R4` darà 4x4=16 campioni per pixel.

Il secondo metodo inizia proiettando quattro raggi agli angoli di ogni pixel. Se il colore risultante differirà di molto dal valore di soglia allora saranno lanciati altri raggi. Questo sarà fatto ricorsivamente, il pixel resterà diviso in quattro sotto-pixel che saranno separatamente analizzati nei seguenti passaggi. Il vantaggio di questo metodo è che riduce il numero di raggi che devono essere tracciati. I campioni che stanno nei pixel e nei sotto-pixel sono immagazzinati e riutilizzati dove conveniente per evitare di dover ritracciare alcuni raggi. La ricorsività di questo metodo lo rende *adattivo* (si parla infatti di *sovracampionamento adattivo*, o *adaptive supersampling*) : si concentrerà infatti su quelle parti di pixel che hanno maggior bisogno di sovracampionamento (vedi figura sotto).

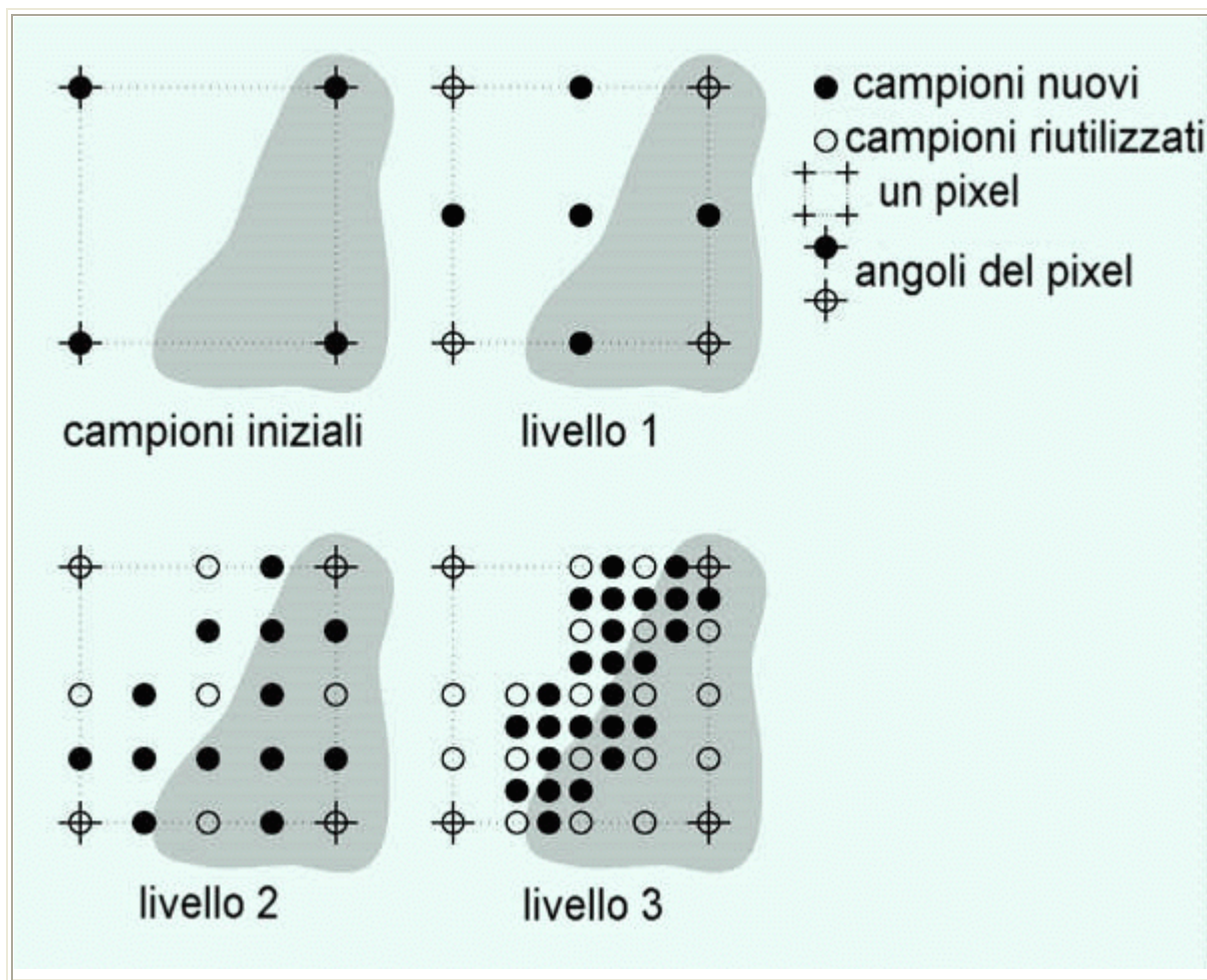



Fig. 196-Un esempio di Funzionamento del Sovracampionamento Adattivo

Il massimo numero di suddivisioni è specificato dall'opzione `Antialias_Depth=n` dal parametro `+Rno`. Questo è diverso per il metodo non adattivo, dove il numero totale del sovracampionamento è specificato. Il numero massimo di suddivisioni risulterà in un numero massimo di sovracampionamenti per pixel e sarà dato come specificato nella seguente tabella.

`+Rn`	numero di campioni per pixel sovracampionato per il metodo *non adattivo*	numero di campioni per pixel sovracampionato per il metodo *adattivo*
1	1	9
2	4	25
3	9	81
4	16	289
5	25	1089
6	36	4225
7	49	16641
8	64	66049
9	81	263169

E' facile notare come il numero massimo di campionamenti rispetto al metodo adattivo è raramente raggiungibile. Se questo metodo è usato senza anti-aliasing sarà determinato dalla media dei raggi

lanciati ai suoi angoli. Nella maggioranza dei casi il livello tre sarà più che sufficiente.

Un altro modo per ridurre le malefatte dell'aliasing è quello di introdurre del "rumore" nel processo di campionamento. Questo metodo è chiamato jittering e funziona perché il nostro meccanismo di vista è molto meno attento al "rumore" che ai colori o ai motivi di colore regolari. La posizione del campionamento è soggetta ad una piccola turbolenza dove si sta facendo uso dell'anti-aliasing. Il jittering è usato di default, ma può essere disattivato con l'opzione `Jitter=off` o con il parametro `-J`. La quantità di jittering può essere impostata con l'opzione `Jitter_Amount=n.n`. Quando si usano i parametri della linea di comando il valore di jittering può essere impostato immediatamente dopo il parametro `+J`. Per esempio `+J0.5` usa la metà del valore normale di jittering. Il valore predefinito di `jitter` è anche il massimo valore di jittering possibile, che assicurerà che tutto il processo di sovracampionamento rimarrà all'interno del pixel su cui si sta lavorando. E' da notare che il jittering è casuale e non ripetibile, quindi dovrai *evitare di usarlo* nelle animazioni poiché il pixel che ha subito il processo di anti-aliasing potrebbe risultare diverso di fotogramma in fotogramma.

Se non viene usato l'anti-aliasing, per ogni pixel viene preso un solo campione indipendentemente dal metodo di sovracampionamento specificato qui sopra.

7. Linguaggio di Descrizione delle Scene

Il linguaggio di descrizione delle scene di POV-Ray consente di descrivere le scene in modo leggibile e comprensibile. I file vengono creati come normali file di testo ASCII usando un qualunque editor. Il nome del file di input è specificato a POV-Ray usando l'opzione `Input_File_Name=file` o il parametro `+Ifile`. Di default i file hanno estensione **.pov**. POV-Ray legge e analizza il file creandosi un modello interno della scena e quindi renderizzandola. La sintassi di un file di una scena conterrà quindi una serie di elementi in qualunque ordine quali :

```
Direttive del liguaggio
camera { oggetti_camera }
Frasi sugli oggetti
Frasi sull'atmosfera
global_settings { Impostazioni }
```

vedere la sezione "Istruzioni del Linguaggio", la sezione "Oggetti", la sezione "Camera", la sezione "Effetti atmosferici" e la sezione "Impostazioni Globali" per le spiegazioni più dettagliate.

7.1 Elementi Fondamentali

Il linguaggio di POV-Ray è composto da identificatori, parole chiave, funzioni, stringhe, simboli speciali e commenti. Il testo di una scena di POV-Ray può essere di un formato qualunque. E' possibile mettere in linee separate o nella stessa linea i vari descrittori, aggiungere linee vuote, spazi senza però dividere parole chiave o identificatori.

7.1.1 Identificatori e Parole Chiave

POV-Ray permette di definire *identificatori* per un uso anche successivo nella scena. Un identificatore può essere lungo da 1 a 40 caratteri, essere composto da lettere minuscole o maiuscole, numeri da 0 a 9 e "sotto-barre" ("_").
POV-Ray ha un certo numero di *parole chiave* riservate che sono elencate qui sotto.

```
aa_level
aa_threshold
abs
acos
```

acosh
adaptive
adc_bailout
agate
agate_turb
all
alpha
ambient
ambient_light
angle
aperture
arc_angle
area_light
asc
asin
asinh
assumed_gamma
atan
atan2
atanh
atmosphere
atmospheric_attenuation
attenuating
average
background
bicubic_patch
black_hole
blob
blue
blur_samples
bounded_by
box
box_mapping
bozo
break
brick
brick_size
brightness
brilliance
bumps
bump_map
bump_size
camera
case
caustics
ceil
checker
chr
clipped_by
clock
color
color_map

```
colour
colour_map
component
composite
concat
cone
confidence
conic_sweep
constant
control0
control1
cos
cosh
count
crackle
crand
cube
cubic
cubic_spline
cylinder
cylindrical_mapping
debug
declare
default
degrees
dents
difference
diffuse
direction
disc
distance
distance_maximum
div
dust
dust_type
eccentricity
else
emitting
end
error
error_bound
exp
exponent
fade_distance
fade_power
falloff
falloff_angle
false
file_exists
filter
finish
```

fisheye
flatness
flip
floor
focal_point
fog
fog_alt
fog_offset
fog_type
frequency
gif
global_settings
glowing
gradient
granite
gray_threshold
green
halo
height_field
hexagon
hf_gray_16
hierarchy
hollow
hypercomplex
if
ifdef
iff
image_map
include
int
interpolate
intersection
inverse
ior
irid
irid_wavelength
jitter
julia_fractal
lambda
lathe
leopard
light_source
linear
linear_spline
linear_sweep
location
log
looks_like
look_at
low_error_factor

mandel
map_type
marble
material_map
matrix
max
max_intersections
max_iteration
max_trace_level
max_value
merge
mesh
metallic
min
minimum_reuse
mod
mortar
nearest_count
no
normal
normal_map
no_shadow
number_of_waves
object
octaves
off
offset
omega
omnimax
on
once
onion
open
orthographic
panoramic
perspective
pgm
phase
phong
phong_size
pi
pigment
pigment_map
planar_mapping
plane
png
point_at
poly
polygon
pot
pow

ppm
precision
prism
pwr
quadratic_spline
quadric
quartic
quaternion
quick_color
quick_colour
quilted
radial
radians
radiosity
radius
rainbow
ramp_wave
rand
range
reciprocal
recursion_limit
red
reflection
refraction
render
repeat
rgb
rgbf
rgbft
rgbt
right
ripples
rotate
roughness
samples
scale
scallop_wave
scattering
seed
shadowless
sin
sine_wave
sinh
sky
sky_sphere
slice
slope_map
smooth
smooth_triangle
sor
specular

```
sphere
spherical_mapping
spiral
spiral1
spiral2
spotlight
spotted
sqr
sqrt
statistics
str
strcmp
strength
strlen
strlwr
strupr
sturm
substr
superellipsoid
switch
sys
samples
scale
scallop_wave
scattering
seed
shadowless
sin
sine_wave
sinh
sky
sky_sphere
slice
slope_map
smooth
smooth_triangle
sor
specular
sphere
spherical_mapping
spiral1
spiral2
spotlight
spotted
sqr
sqrt
statistics
str
strcmp
strength
strlen
strlwr
```

strupr
sturm
substr
superellipsoid
switch
sys

t
tan
tanh
text
texture
texture_map
tga
thickness
threshold
tightness
tile2
tiles
torus
transform
translate
transmit
triangle
triangle_wave
true
ttf
turbulence
turb_depth
type

u
ultra_wide_angle
union
up
use_color
use_colour
use_index
u_steps

v
val
variance
vaxis_rotate
vcross
vdot
version
vlength
vnormalize
vrotate
v_steps

warning
warp
water_level

```
waves
while
width
wood
wrinkles
x
y
yes
z
```

Tutte le parole riservate sono completamente minuscole e quindi è preferibile che i tuoi identificatori contengano almeno una lettera maiuscola per evitare di incorrere in conflitti con le parole riservate.
Le seguenti parole non sono più utilizzate da POV-Ray ma rimangono comunque riservate.

```
bumpy1
bumpy2
bumpy3
incidence
pattern1
pattern2
pattern3
spiral
test_camera_1
test_camera_2
test_camera_3
test_camera_4
track
volume_object
volume_rendered
vol_with_light
```

7.1.2 Commenti

I commenti sono porzioni di testo incluse nel file di scena che consentono una lettura più comprensibile del file stesso. Sono ignorati dal ray-tracer e sono scritti solo per tua informazione. Ci sono due tipi di commenti in POV-Ray.
Due barre inclinate sono usate per commenti di una singola linea. Tutto quello che viene dopo una doppia barra ("//") è ignorato dal ray-tracer, per esempio :

```
// Questa linea è ignorata
```

Puoi trovare informazioni sulla linea del file di scena di fronte al commento in questo modo:

```
oggetto { FooBar} // questo è un oggetto
```

L'altro tipo di commento è usato per escludere più linee di testo dall'analisi. Comincia con " /* " e finisce con " */ ". Tutto ciò che si trova in mezzo è ignorato. Per esempio:

```
/* Queste linee
sono ignorate
dal
ray-tracer */
```

Questo può essere utile se vuoi temporaneamente rimuovere elementi da un file di scena. Questi commenti " /*.... */ " possono essere contenuti nei commenti caratterizzati dalle doppie barre inclinate // ; i commenti così possono essere usati temporaneamente o permanentemente all'interno di una scena. /*.... */ i commenti possono anche essere annidati, ad esempio è consentita la seguente espressione :

```
/* Questo è un commento
// Questo anche
/* Questo pure */
*/
```

Usa i commenti con estrema libertà. Bene utilizzati migliorano realmente la leggibilità dei file di scena.

7.1.3 Espressioni Decimali

Molte parti del linguaggio di POV-Ray richiedono di specificare uno o più numeri decimali. Numeri decimali possono essere specificati usando lettere dell'alfabeto, identificatori o funzioni che restituiscono valori decimali. Puoi creare anche espressioni decimali molto complesse da combinazioni di ognuno di questi usando vari operatori comuni.
Se POV-Ray richiede un numero intero, potrai inserire anche un numero decimale che però verrà troncato e trasformato in numero intero. Quando POV-Ray ha bisogno di un valore logico o booleano, esso interpreta ogni numero decimale diverso da zero come **vero** e ogni numero uguale a zero come **falso**. Poiché il confronto di numeri decimali è soggetto ad errori di arrotondamento, POV-Ray considera valori estremamente vicini a zero come **falsi** quando esegue funzioni booleane. Normalmente entità numeriche i cui valori assoluti sono meno di un valore epsilon preimpostato (tolleranza) sono considerate **false** quando usate in espressioni logiche. Il valore di epsilon è dipendente dal sistema ma generalmente è di circa 1.0e-10. Due valori decimali a e b sono considerati uguali se il valore assoluto di (a-b) risulta minore di epsilon.

7.1.3.1 Costanti Decimali

Valori decimali rappresentati da lettere sono indicati da un segno opzionale ("+" o "-"), un punto decimale opzionale e più cifre. Se il numero è un intero puoi omettere il punto decimale seguito da zero. Se esso è espresso solo mediante la parte decimale, puoi omettere lo zero. POV-Ray supporta la comune notazione scientifica per numeri molto grandi o molto piccoli. I numeri seguenti sono tutti valori letterali decimali validi:

```
- 2.0 - 4 34 3.4e6 2e-5 .3 0.6
```

7.1.3.2 Identificatori Decimali

Gli identificatori decimali possono essere dichiarati per rendere più leggibile un file di scena e per parametrizzare una scena così che cambiando una singola dichiarazione cambino molti valori. Un identificatore è dichiarato come segue.

```
#declare IDENTIFICATORE = ESPRESSIONE
```

Dove `IDENTIFICATORE` è il nome dell'identificatore e può essere lungo fino a 40 caratteri ed `ESPRESSIONE` è ogni espressione decimale considerata valida. Qui di seguito alcuni esempi.

```
#declare Count = 0
#declare Rows = 5.3
```

```
#declare Cols = 6.15
#declare Number = Rows*Cols
#declare Count = Count+1
```

Come si vede nell'ultimo esempio, puoi ri-dichiarare un identificatore decimale e puoi riutilizzare valori dichiarati precedentemente. Ci sono molti identificatori già predefiniti che POV-Ray dichiara per te. Vedi "Identificatori Incorporati" per dettagli.

7.1.3.3 Operatori Decimali

Espressioni aritmetiche decimali possono essere create da costanti decimali, identificatori o funzioni usando gli operatori seguenti in questo ordine di precedenza....

(...)	espressioni tra parentesi
+A -A !A	più, meno, e "NOT" logico
A*B A/B	moltiplicazione e divisione
A+B A-B	somma e sottrazione

Possono anche essere create espressioni relazionali, logiche e condizionali. Esiste comunque una restrizione e cioè che questi tipi di espressione devono essere inclusi in parentesi. Questa restrizione, che non è imposta dalla maggior parte dei linguaggi di programmazione, è necessaria perché POV-Ray consente di unire espressioni decimali e vettoriali. Senza le parentesi c'è un problema di ambiguità. Le parentesi non sono richieste per l'operatore logico **not** "!" mostrato prima. Gli operatori e le loro priorità sono mostrati qui di seguito.

Espressioni relazionali: Gli operandi sono espressioni dell'aritmetica ed il risultato è sempre booleano con 1 per vero e 0 per falso. Tutti gli operatori relazionali hanno la stessa precedenza.

(A< B)	A è minore di B
(A<= B)	A è minore o uguale a B
(A= B)	A è uguale a B (ovvero abs(A-B)< EPSILON)
(A!= B)	A non è uguale a B (ovvero abs(A-B)>= EPSILON)
(A>= B)	A è più grande o uguale a B
(A> B)	A è più grande di B

Espressioni logiche: Gli operandi sono convertiti a valori booleani di 0 per falso e 1 per vero. Il risultato è sempre booleano. Tutti gli operatori logici hanno la stessa precedenza.

(A & B)	vero solo se sia A che B sono veri, falso altrimenti
(A | B)	falso se entrambi sono falsi, vero in tutti gli altri casi

Espressioni condizionali: L'operando C è booleano mentre gli operandi A e B rappresentano espressioni qualsiasi . Il risultato è dello stesso tipo di A e B.

(C ? A : B)	se C allora A altrimenti B

Assumendo i vari identificatori che sono stati dichiarati, i seguenti esempi rappresentano espressioni valide...

```
1+2+3 2*5 1/3 Row*3 Col*5
(Offset-5)/2 Questo/Quello+Altro*Cosa
((Questo<Quello) & (Altro>=Cosa)?Foo:Bar)
```

Le espressioni sono valutate nel seguente ordine : prima con parentesi da sinistra a destra a partire dalle più interne , poi + , - o ! , poi prodotti e divisioni, poi addizioni e sottrazioni, poi espressioni relazionali, poi logiche, poi condizionali.

7.1.4 Espressioni Vettoriali

POV-Ray spesso richiede di specificare un vettore. Un vettore è un insieme di valori decimali relazionati. I vettori possono essere specificati usando lettere, identificatori o funzioni che restituiscono un valore vettoriale. Puoi creare anche espressioni vettoriali molto complesse da combinazioni di ognuno di questi usando vari operatori comuni.
I vettori di POV-Ray possono avere da due a cinque componenti ma la maggioranza dei vettori ha tre componenti. A meno che tu non abbia specificato altrimenti, devi assumere che la parola che rappresenta il vettore abbia tre componenti. POV-Ray opera in un sistema di coordinate a tre dimensioni XYZ ed userà un vettore a tre componenti per specificare i valori di x, y e z . In alcuni casi POV-Ray ha bisogno solo di due coordinate. Queste sono specificate spesso da un vettore 2D chiamato vettore UV . Gli oggetti frattali usano vettori 4D. Espressioni del colore usano vettori 5D ma ti permettono di specificare 3, 4 o 5 componenti e di tralasciare i valori di default per i componenti non specificati. Matematicamente, i vettori con 2, 4 o 5 componenti sono trattati nello stesso modo dei vettori 3D. Solo, hanno un numero diverso di componenti.

7.1.4.1 Costanti Vettoriali

I vettori consistono di due, tre, quattro o cinque espressioni messe fra parentesi angolari < e > . I termini sono separati da virgole. Nell'esempio qui riportato c'è un tipico vettore a tre componenti:

```
< 1.0, 3.2,- 5.4578>
```

Le virgole tra le componenti sono necessarie per evitare che il programma consideri il secondo termine come l'espressione decimale 3.2-5.4578 e che non esiste il 3° termine. Se vedi un messaggio di errore che segnala la mancanza di un valore decimale, avendo invece trovato ' > ' probabilmente hai omesso una virgola.
Qualche volta POV-Ray richiede di specificare valori decimali e vettori affiancati. Le regole per espressioni vettoriali permettono di unire vettori con vettori o vettori con valori decimali, così sono richieste virgole di separazione ogni qualvolta potrebbe sorgere ambiguità. Per esempio< 1,2,3> -4 è valutato come un valore decimale unito ad un'espressione di vettore dove 4 è sottratto da ciascuno dei componenti e quindi ha come conseguenza<- 3,- 2, -1>. La virgola in <1,2,3>, 4 significa che questo è un vettore seguito da un valore decimale.
Ogni componente può essere un'espressione decimale. Per esempio

```
< Questo+3, Quello/ 3, 5*Altra-Cosa>
```

è un vettore valido.

7.1.4.2 Identificatori Vettoriali

Gli identificatori del vettore possono essere dichiarati per rendere un file di scena più leggibile e ottenere scene parametrizzate così che cambiando una singola dichiarazione, cambino molti valori. Un identificatore è dichiarato come segue.

```
#declare IDENTIFICATORE = ESPRESSIONE
```

Dove `IDENTIFICATORE` è il nome dell'identificatore che può essere lungo fino a 40 caratteri ed `ESPRESSIONE` è ogni espressione valida che esprima il valore del vettore. Qui di seguito alcuni esempi.

```
# declare Qui = < 1,2,3 >
# declare Là = < 3,4,5 >
# declare Salta = < Foo* 2, Bar-1, Bob/ 3 >
# declare Strada = Là - Qui
# declare Salta = Salta + < 1,2,3 >
```

Nota che puoi utilizzare un identificatore di vettore per usare il suo nome senza le parentesi angolate. Come mostrano gli ultimi esempi, puoi ri-dichiarare un identificatore vettore e puoi usare valori dichiarati preventivamente in quest'ultima dichiarazione. Ci sono molti identificatori "incorporati" che POV-Ray dichiara per te. Vedi "Identificatori Incorporati" per maggiori dettagli.

7.1.4.3 Operatori Vettoriali

Costanti vettoriali, identificatori e funzioni possono essere associati in espressioni nello stesso modo in cui avviene per valori decimali. Le operazioni vengono eseguite componente per componente. Per esempio < 1, 2, 3 > + < 4, 5, 6 > viene valutato come < 1+4, 2+5, 3+6 > oppure < 5, 7, 9 >. Altre operazioni sono eseguite in modo simile. Per esempio (< 1, 2, 3 >=< 3, 2, 1 >) è valutato come < 0, 1, 0 > poiché i componenti centrali sono uguali mentre gli altri non lo sono. Ammettiamo che questo non è molto utile ma può essere interessante per altre operazioni sui vettori.
Un'espressione condizionale come (C? A : B) richiede che C sia un'espressione decimale ma A e B possono essere espressioni vettoriali. Il risultato è che l'intera espressione condizionale viene valutata come un vettore valido. Per esempio se Foo e Bar sono valori decimali allora

```
Foo < Bar ? < 1,2,3 >: < 5,6,7 >
```

viene valutato il vettore < 1,2,3 > se Foo è minore di Bar, o < 5,6,7 > in caso contrario.
Puoi usare l'operatore *punto* per estrarre un singolo componente da un vettore. Supponi che l'identificatore **Spot** fosse stato precedentemente definito come un vettore. Allora **Spot.x** è un valore decimale che rappresenta il primo componente di questo vettore a tre componenti. Similmente **Spot.y** e **Spot.z** rappresentano il 2° e il 3° componente. Se Spot era invece un vettore UV a due componenti, avresti dovuto usare **Spot.u** e **Spot.v** per estrarre il primo e secondo componente. Per un vettore 4D usa **.x**, **.y**, **.z** e **.t** per estrarre ciascun componente decimale. L'operatore punto è usato anche in espressioni del colore che sono prese in considerazione più avanti.

7.1.4.4 Promozione di Operatori

Puoi usare *una sola* espressione decimale per definire un vettore i cui componenti sono dello stesso tipo. POV-Ray sa quando si necessita di un vettore di un particolare tipo e promuoverà un valore decimale se ciò è necessario. Per esempio la frase "`scale`" di POV-Ray richiede un vettore a tre componenti. Se si specifica `scale 5` allora POV-Ray lo interpreta come `scale < 5,5,5 >` ovvero l'intenzione di voler scalare di 5 in ogni direzione.

Le versioni precedenti di POV-Ray 3.0 consentivano tale uso del valore decimale come vettore in ambiti ristretti come con `scale` e `turbulence`. Adesso puoi usare questa scorciatoia ovunque. Per esempio...

```
box{ 0,1} // Questo è lo stesso di box {< 0,0,0 >,< 1,1,1 >}
sphere { 0,1}// Questo è lo stesso di sphere {< 0,0,0>, 1}
```

Quando viene promosso un valore decimale in un vettore di 2, 3, 4 o 5 componenti, a tutti i componenti viene assegnato il valore decimale, invece quando si promuove un vettore con un numero più basso di componenti in un vettore di ordine più alto, tutti i componenti rimanenti sono posti a zero. Per esempio se POV-Ray aspetta un vettore 4D e si specifica 9, il risultato è < 9,9,9,9 > ma se si specifica < 7,6 > il risultato è < 7,6,0,0 >.

7.1.5 Specificare i Colori

POV-Ray spesso richiede di specificare un colore. I colori consistono di cinque valori o *componenti del colore*. I primi tre sono il rosso, il verde e il blu. Essi specificano l'intensità dei colori primari rosso, verde e blu usando un sistema additivo del colore come quello usato dai fosfori rossi, verdi e blu per la visualizzazione del colore sul monitor.
Il 4° componente, chiamato *filtro*, specifica l'ammontare della *trasparenza filtrata* di una sostanza. Alcuni esempi della vita reale sulla trasparenza filtrata possono essere il vetro colorato di una vetrata, o del cellophane colorato. La luce passando attraverso un oggetto non opaco assume il suo colore nello stesso modo in cui un oggetto assorbe alcune frequenze di luce mentre consente ad altre di attraversarlo. Il colore dell'oggetto è sottratto dalla luce che lo attraversa e ciò prende il nome di *trasparenza sottrattiva*.
Il 5° componente, chiamato *emettitore* o *trasmittanza*, specifica l'ammontare di luce non filtrata che passa attraverso una superficie. Alcuni esempi dall'esperienza reale sulla *trasparenza non filtrata* possono essere rappresentati da una stoffa a trama fine, che consente di vederle attraverso, una maglia a rete, o uno strato uniforme di polvere su una superficie. In questi esempi, tutte le frequenze di luce possono passare attraverso i microscopici buchi della superficie. Benché l'ammontare di luce che attraversa la superficie sia diminuito, il colore della luce che attraversa rimane immutato. Il colore dell'oggetto è aggiunto alla luce che lo attraversa quindi questo tipo di trasparenza è chiamato *trasparenza additiva*.
Nota che le prime versioni di POV-Ray usavano la parola chiave `alpha` per specificare la trasparenza filtrata. Questo termine è spesso usato per descrivere la trasparenza non filtrata. Per questa ragione la parola chiave `alpha` non viene più usata.
Ciascuna delle cinque componenti di un colore è un valore decimale che normalmente ha un'oscillazione compresa tra 0.0 e 1.0. Comunque alcuni valori, anche negativi possono essere usati. I colori possono essere specificati usando vettori, parole chiave con valori decimali o identificatori. Puoi creare anche espressioni del colore molto complesse da combinazioni di ognuno di questi operatori comuni. La sintassi per specificare un colore si è evoluta in POV-Ray dalla prima versione del programma. Abbiamo mantenuto la sintassi originale della parola chiave di base e aggiunto una scorciatoia alla notazione di vettore. Entrambe le sintassi, vecchia e nuova, sono accettabili benché la sintassi del vettore sia più facile da usare quando si creano espressioni di colore.

7.1.5.1 Vettori Colore

La sintassi per un vettore colore è una delle seguenti....

```
color rgb   vettore3
color rgbf   vettore4
color rgbt   vettore4
color rgbft   vettore5
```

dove `vettore3`, `vettore4`o `vettore5` rappresentano ogni espressione valida di un vettore da 3, 4 o 5 componenti. Per esempio:

```
color rgb < 1.0, 0.5, 0.2 >
```

specifica un colore il cui componente rosso è 1,0 o il 100% della massima intensità. Il componente verde è 0,5 o il 50% della massima intensità ed il componente blu è 0,2 o il 20% della massima intensità. Benché i componenti del filtro e della trasmittanza non siano specificati, esistono e si sottintendono posti al loro valore di default, ovvero uguali a zero (nessuna trasparenza).
La parola chiave `rgbf` richiede un quarto componente del vettore. Il quarto componente è il componente del *filtro* mentre il componente *trasmittanza* assume per default il valore zero.
Similmente la parola chiave `rgbt` richiede quattro componenti dove il 4° valore è assegnato al 5° componente che è la trasmittanza ed il componente del filtro è posto uguale a zero.
La parola chiave `rgbft` permette di specificare tutti i cinque componenti. Internamente alle espressioni tutti i cinque componenti sono sempre usati.
Nella maggior parte delle circostanze la parola chiave `color` è opzionale e può essere omessa.
Viene supportata anche l'espressione inglese o canadese "`colour`". In alcune circostanze, se l'espressione del vettore è un'espressione a 5 componenti o c'è un identificatore del colore nell'espressione, allora la parola chiave `rgbft` è opzionale.

7.1.5.2 Parole Chiave

Il vecchio metodo della parola chiave per specificare un colore è ancora utile e molti utenti lo preferiscono. Come un vettore colore, inizia con la parola chiave opzionale `color`. Questa è seguito da ognuna delle cinque parole chiave addizionali `red, green, blue, filter` e `transmit` (rosso, verde, blu, filtro e trasmittanza). Ciascuna di queste parole chiave del componente è seguita da un'espressione decimale del valore. Per esempio

```
color red 1.0 green 0.5
```

Specifica un colore il cui componente rosso è 1,0 o 100% della massima intensità ed il componente verde è 0,5 o 50% della massima intensità. Benché i componenti del blu, del filtro ed dell'emettitore non siano specificati, esistono e sono posti al loro valore di default, ovvero a zero. Le parole chiave del componente possono essere date in qualsiasi ordine e se un componente non è specificato il suo valore assume per default lo zero.

7.1.5.3 Identificatori di Colore

Gli identificatori del colore possono essere dichiarati per rendere il file della scena più leggibile e per parametrizzare le scene, così che cambiando una singola dichiarazione cambino molti valori. Un identificatore del colore è espresso come l'uno o l'altro caso del seguente esempio...

```
#declare IDENTIFICATORE = VETTORE_COLORE
#declare IDENTIFICATORE = PAROLE_CHIAVE_COLORE...
```

Dove `IDENTIFICATORE` è il nome dell'identificatore, lungo fino a 40 caratteri e `VETTORE_COLORE` o `PAROLE_CHIAVE_COLORE` sono specificazioni del colore valide, come viene descritto nelle due sezioni precedenti di questo documento. Di seguito alcuni esempi....

```
#declare White = rgb < 1,1,1 >
#declare Cyan = color blue 1.0 green 1.0
```

```
#declare Weird = rgb <Foo*2,Bar-1,Bob/3>
#declare LightGray = White*0.8
#declare LightCyan = Cyan red 0.6
```

Come l'esempio `LightGray` mostra, non hai bisogno di alcuna parola chiave del colore quando crei espressioni di colore basate su precedenti dichiarazioni. L'ultimo esempio mostra che puoi utilizzare un identificatore del colore con la sintassi della parola chiave. Accertati che l'identificatore venga prima di ogni altro componente di parole chiave.
Come accade per valori decimali e vettori, puoi ri-definire i colori lungo tutta una scena, ma tale necessità è rara.

7.1.5.4 Operatori sul Colore

I vettori colore possono essere combinati in espressioni allo stesso modo in cui avviene per valori decimali o per valori vettoriali. Le operazioni sono compiute componente per componente. Per esempio

```
rgb < 1.0, 0.5 0.2>* 0.9
```

viene valutato nello stesso modo di

```
rgb < 1.0, 0.5 0.2 > * < 0.9, 0.9, 0.9 >
```

o di

```
rgb< 0.9, 0.45, 0.18 >
```

Altre operazioni sono effettuate sullo stesso principio, "componente per componente". Puoi usare l'operatore punto per estrarre un singolo componente da un colore. Supponi che l'identificatore `Shade` fosse stato definito precedentemente come un colore. Allora `Shade.red` è il valore decimale del componente rosso di `Shade`. Similmente `Shade.green`, `Shade.blue`, `Shade.filter` e `Shade.transmit` estraggono il valore decimale degli altri componenti del colore.

7.1.5.5 Comuni Tranelli sul Colore

La varietà e la complessità dei metodi relativi alla specificazione del colore possono condurre ad alcuni comuni errori. Vi sono alcune cose da considerare quando si specifica un colore.
Quando si usa un valore di filtro, i colori che lo attraversano vengono moltiplicati per le componenti del colore a cui si riferisce il filtro. Per esempio, se la luce è grigia (rgb < 0.9, 0.9, 0.9 >) il passaggio attraverso un filtro rgbf < 1.0, 0.5, 0.0, 1.0 > darà come risultato il colore rgb < 0.9, 0.45, 0.0 > dove la componente rossa non viene modificata, il valore della componente verde viene dimezzato passando da 0,9 a 0,45 ed il blu risulta totalmente bloccato. Spesso si pensa di definire un oggetto completamente trasparente con la frase

```
color filter 1.0
```

ma questo comporta un valore uguale a zero per le componenti di rosso, verde e blu. In questo modo si specifica, infatti, un filtro totalmente nero che impedisce il passaggio della luce. La procedura corretta è descritta dagli esempi che seguono:

```
color red 1.0 green 1.0 blue 1.0 filter 1.0
```

oppure

```
color transmit 1.0
```

Nel secondo esempio i valori di rgb non sono importanti. Tutte le componenti della luce non verranno modificate da questo filtro.
Un altro errore comune è l'uso di identificatori di colore insieme alle parole chiavi relative al colore. Per esempio:

```
color Mio_Colore red 0.5
```

sostituisce qualunque valore della componente rossa in Mio_Colore con una componente rossa di 0.5.

```
color Mio_Colore + red 0.5
```

invece, aggiunge 0.5 alla componente rossa di Mio_Colore e chiaramente....

```
color Mio_Colore * red 0.5
```

dimezza la componente rossa, come era prevedibile, ma moltiplica anche le componenti verde, blu, filtro e quella della trasparenza per zero! Dopo questa moltiplicazione il risultato dell'espressione è rgbft < 0.5, 0, 0, 0, 0 >.
Nell'esempio seguente i risultati non modificano Mio_Colore.

```
color red 0.5 Mio_Colore
```

Questo perché l'identificatore sovrascrive completamente il valore precedente. Quando si usano identificatori con parole chiave del colore, l'identificatore dovrebbe essere al primo posto.
Infine, alcune sintassi di POV-Ray consentono specificazioni del colore, ma usano solo la parte rgb. In questi casi è consentito usare un valore decimale là dove un colore ne abbia bisogno. Per esempio:

```
finish { ambient 1 }
```

La parola chiave `ambient` richiede la specificazione di un colore, così il valore 1 diventa il vettore < 1, 1, 1, 1, 1 >.
Anche

```
pigment { color 0.4 }
```

è consentito, ma potrebbe essere diverso da ciò che intendi. Lo 0.4 diventa < 0.4, 0.4, 0.4, 0.4, 0.4 > col filtro e la trasparenza impostati a 0.4. E' più probabile che tu volessi...

```
pigment { color rgb 0.4 }
```

in questo caso ci si aspetta un vettore a 3 componenti. Perciò lo 0.4 è promosso a < 0.4, 0.4, 0.4, 0.0, 0.0 > con un valore pari a zero sia per il filtro che per la trasparenza.

7.1.6 Stringhe

Il linguaggio di POV-Ray richiede di specificare una stringa di caratteri da usare per indicare un nome di file, un testo da inserire nei messaggi che POV-Ray visualizza durante i rendering o come oggetto di una scena. Le stringhe possono essere specificate usando lettere, identificatori o funzioni che restituiscono i valori della stringa. Benché non sia possibile formulare espressioni contenenti operatori simbolici così come avviene per i valori decimali, i vettori o i colori, puoi compiere varie operazioni sulle stringhe usando funzioni appropriate. Alcune applicazioni delle stringhe in POV-Ray permettono di inserire caratteri di formattazione, ma non di stampa come ad esempio i caratteri "a capo" o "avanzamento modulo".

7.1.6.1 Costanti Stringa

Le costanti stringa sono racchiuse tra virgolette (") e sono composte da un massimo di 256 caratteri ASCII stampabili. Le seguenti rappresentano tutte costanti di stringa valide:

"Qui" "Là" "myfile.gif" "textures.inc"

7.1.6.2 Identificatori di Stringa

Gli identificatori di stringa possono essere dichiarati per rendere i file della scena maggiormente leggibili e per parametrizzare le scene così che, cambiando una singola dichiarazione, cambino molti valori. Un identificatore è dichiarato come segue.....

```
# declare IDENTICATORE= STRINGA
```

Dove IDENTIFICATORE è il nome dell'identificatore che può arrivare fino a 40 caratteri. STRINGA è una costante stringa, un identificatore di stringa o una funzione che riporta un valore di stringa. Di seguito alcuni esempi.

```
#declare Nome_Carattere = "arial.ttf"
#declare Inc_File = "miofile.inc"
#declare Nome = "John"
#declare Nome = concat(Nome," Doe")
```

Come mostra l'ultimo esempio, puoi ri-dichiarare un identificatore di stringa e puoi usare valori precedentemente dichiarati in quella nuova dichiarazione.

7.1.7 Identificatori Incorporati

Vi sono molti identificatori decimali e vettore incorporati in POV-Ray. Puoi usarli per specificare valori o per creare espressioni, ma non è possibile modificare i loro valori.

7.1.7.1 Costanti Numeriche Incorporate

La maggior parte degli identificatori incorporati non cambiano mai valore. Essi sono definiti come se le seguenti linee fossero scritte all'inizio di ogni scena.

```
#declare pi = 3.1415926535897932384626
#declare true = 1
#declare yes = 1
#declare on = 1
#declare false = 0
#declare no = 0
#declare off = 0
#declare u = <1,0>
#declare v = <0,1>
```

```
#declare x = <1,0,0>
#declare y = <0,1,0>
#declare z = <0,0,1>
#declare t = <0,0,0,1>
```

L'identificatore incorporato decimale `pi` (π) risulta evidentemente utile in espressioni di matematica che prendono in considerazione dei cerchi.
Gli identificatori incorporati decimali `on`, `off`, `yes`, `no`, `true` e `false` sono destinati all'uso di costanti booleane.
Gli identificatori incorporati del vettore x, y e z consentono una maggiore leggibilità del tuo file di scena quando sono usati in espressioni di vettore. Per esempio.

```
plane { y, 1} // Il vettore normale è evidentemente "y".
plane { <0, 1, 0 >, 1} // Questo è più difficile da leggere.

translate 5*x // Muove 5 unità nella direzione "x".
translate < 5, 0, 0 > //Questo è meno ovvio.
```

Un espressione come 5*x equivale a 5< 1, 0, 0 > o < 5, 0, 0 >.
Allo stesso modo per i vettori 2D si possono usare `u` e `v`. Quando si usano dei vettori 4D, questi dovrebbero chiamarsi `x`, `y`, `z` e `t`.

7.1.7.2 Identificatore Clock

L'identificatore predefinito decimale clock è usato per controllare le animazioni in POV-Ray. Diversamente da alcuni programmi di animazione, le scene animate in POV-Ray non dipendono dal numero (intero) del fotogramma. Invece, dovrai progettare le tue scene basandoti sull'identificatore decimale clock. Per scene non animate il suo valore per default è pari a 0, ma è possibile assegnargli qualsiasi valore decimale usando l'opzione del file .INI Clock= n.n o il parametro +Kn.n.
Altre opzioni e corrispondenti impostazioni dei file .INI possono essere usate per animare automaticamente le scene attraverso il rendering di immagini successive che usano diversi valori di clock. Per default, il valore di clock è 0 per il primo fotogramma e 1 per l'ultimo. Tutte le altre immagini sono inserite tra questi valori. Per esempio se il tuo oggetto deve ruotare di un giro nel corso dell'animazione si dovrebbe specificare: `rotate 360*clock*y`. Allora, mentre clock procede da 0 a 1, l'oggetto ruota intorno all'asse y da 0 a 360 gradi.
Benché il valore di clock cambi da un fotogramma all'altro, non cambierà mai nel corso dell'analisi (parsing) di una scena.
Vedi "Opzioni sull'Animazione" per maggiori dettagli.

7.1.7.3 Identificatore Version

L'identificatore decimale `version` contiene l'impostazione corrente delle opzioni compatibili con la versione del programma specificata. Benché questo assuma per default il valore 3, che è il numero corrente della versione di POV-Ray, il valore iniziale della versione potrebbe essere introdotto nell'opzione del file .INI Version= n.n o dal comando di linea +MVn.n. Questo fa analizzare a POV-Ray la scena usando la sintassi di una versione precedente.
L'opzione .INI ha effetto solo nell'impostazione iniziale. Diversamente da altri identificatori incorporati, puoi cambiare il valore della versione in tutto il file di una scena. Per cambiare il suo valore usa `#version` e non `#declare`. Tali cambi possono avvenire più volte nei file di scena.
Insieme con l'identificatore `version`, l'istruzione `#version` consente di salvare e ripristinare i valori precedenti di queste impostazioni. Per esempio, si supponga che **mystuff.inc** sia stato creato con POV-Ray 1. All'inizio del file si potrebbe introdurre:

```
#declare Temp_Vers = version // Salva il valore precedente
#version 1.0 // Cambia secondo la modalità della versione 1.0

// Il file compatibile con la versione 1.0 inizia da qui.....

#version Temp_Vers // Ripristina il numero di versione precedente
```

7.1.8 Funzioni

POV-Ray definisce una varietà di funzioni predefinite per manipolare decimali, vettori e stringhe. Le funzioni sono raggruppate secondo il loro uso e non secondo il tipo di valore che calcolano. Per esempio, `vdot` calcola il prodotto scalare (*dot product*) di due vettori ed è elencato tra le funzioni vettoriali sebbene produca come risultato un singolo valore decimale.
La funzione è rappresentata da una parola chiave che specifica il nome della funzione, seguita da una lista di parametri chiusi tra parentesi. I parametri sono separati da virgole. Per esempio:

```
parola_chiave (param1, param2)
```

Le funzioni restituiscono valori che possono essere decimali, vettori o stringhe e possono essere usate in qualunque espressione o dichiarazione che accetti come legittimi quei parametri o identificatori.

7.1.8.1 Funzioni Decimali

Le seguenti funzioni richiedono uno o più parametri decimali e danno come risultato valori decimali. Ipotizziamo che A e B rappresentino ogni espressione valida che dia come risultato un valore decimale. Vedi "Funzioni Vettoriali" e "Funzioni Stringa" per altre funzioni che danno come risultato valori decimali ma il cui input è costituito da vettori e da stringhe.

abs(A): Valore assoluto di A. Se A è negativo, riporta -A in caso contrario riporta A.

acos(A): Arco-coseno di A. Dà come risultato l'angolo, misurato in radianti il cui coseno è A.

asin(A): Arco-seno di A. Dà come risultato l'angolo, misurato in radianti, il cui seno è A.

atan2 (A,B): Arco-tangente di (A/B). Dà come risultato l'angolo, misurato in radianti, la cui tangente è (A/B).
Ritorna un valore appropriato anche se B è uguale a zero. Usa atan2 (A,1) per calcolare la normale funzione atan(A).

ceil(A): Limite massimo di A. Dà come risultato il più piccolo numero intero maggiore di A. Viene arrotondato al numero intero successivo più alto.

cos(A): Coseno di A. Dà come risultato il coseno dell'angolo A, dove A è misurato in radianti.

degrees(A): Converte i radianti in gradi. Dà come risultato l'angolo misurato in gradi il cui valore in radianti è A. La formula è: gradi=A/pi*180.0.

div(A,B): Divisione intera. La parte intera di (A/B).

exp(A): Esponenziale di A. Dà come risultato il valore di *e* elevato alla potenza di A, dove *e* rappresenta il numero irrazionale approssimativamente uguale a 2.71828182846 che è la base dei logaritmi naturali.

floor(A): Limite inferiore di A. Dà come risultato il più alto numero intero minore di A. Arrotonda inferiormente al successivo numero intero più basso.

int(A): La parte intera di A. Dà come risultato la parte intera di A. Arrotonda verso lo zero.

log(A): Logaritmo naturale di A. Dà come risultato il logaritmo naturale di base *e* del valore A dove *e* rappresenta il numero irrazionale approssimativamente uguale a 2.71828182846 che è la base dei logaritmi naturali

max(A,B): Il valore massimo tra A e B. Dà come risultato A se A è maggiore di B. In caso contrario dà B.

min(A,B): Il valore minimo tra A e B. Dà come risultato A se A è più piccolo di B. In caso contrario dà B.

mod(A,B): Valore di A modulo B. Dà come risultato il resto dopo la divisione di A/B. La formula è mod=((A/B)-int(A/B))*B.

pow(A,B): Potenza. Dà come risultato il valore di A elevato alla B-esima potenza.

radians(A): Converte i gradi in radianti. Dà come risultato l'angolo misurato in radianti il cui valore in gradi è rappresentato da A. La formula è: radianti= A*pi/180.0.

rand(A): Dà come risultato il successivo numero pseudo-casuale dal flusso specificato con il numero intero positivo A. Devi richiamare seed() per inizializzare il flusso casuale prima di richiamare rand(). I numeri sono distribuiti uniformemente, e hanno valori compresi tra 0.0 e 1.0, (inclusi entrambi). I numeri generati da flussi separati sono variabili indipendenti casuali.

seed(A): Inizializza un nuovo flusso pseudo-casuale col valore iniziale A come origine. E' riportato il numero corrispondente a questo flusso casuale. Qualsiasi numero di flussi pseudo-casuali può essere usato come viene mostrato nell'esempio di seguito:

```
#declare R1 = seed(0)
#declare R2 = seed(12345)
#sphere { <rand(R1), rand(R1), rand(R1)>, rand(R2) }
```

L'uso di più generatori casuali è molto comodo in situazioni dove venga usato `rand(...)` per posizionare un gruppo di oggetti e poi si decida di usarlo di nuovo in una posizione, magari precedente, per definire colori o altri gruppi di oggetti. Senza flussi separati di `rand(...)`, tutti gli oggetti cambierebbero la loro posizione ogni volta che la funzione viene richiamata. Questo sarebbe molto seccante.

sin(A): Seno di A. Dà come risultato il seno dell'angolo A, dove A è misurato in radianti.

sqrt(A): Radice quadrata di A. Dà come risultato il valore il cui quadrato è A.

tan(A): Tangente di A. Dà come risultato la tangente dell'angolo A, dove A è misurato in radianti.

7.1.8.2 Funzioni Vettoriali

Le seguenti sono funzioni che considerano uno o più vettori e/o parametri decimali e danno come risultato valori vettoriali o decimali. Tutte queste funzioni supportano solo vettori a tre componenti. Definiamo con A e B due espressioni valide che rappresentino due vettori a tre componenti e con F un'espressione valida che rappresenta un numero decimale.

vaxis_rotate(A,B,F): Ruota A attorno a B del valore F. Specificate le coordinate x, y, z del punto nello spazio designato dal vettore A, ruota quel punto intorno ad un asse arbitrario definito dal vettore B. Tale vettore viene ruotato di un angolo specificato in gradi attraverso il valore decimale F. Il risultato è un vettore che contiene le nuove coordinate x,y,z del punto.

vcross(A,B): Prodotto vettoriale di A e B. Dà come risultato il prodotto vettoriale tra i due vettori. Il vettore risultante è perpendicolare ai due vettori originali e la sua lunghezza è proporzionale all'angolo tra essi compreso. Vedi la scena dimostrativa animata **VECT2.POV** come esempio.

vdot(A,B): prodotto scalare di A e B. Dà come risultato un valore decimale che è il prodotto scalare di A con B. La formula è vdot=A.x*B.x + A.y*B.y + A.z*B.z. Vedi la scena dimostrativa animata **VECT2.POV** come esempio.

.

vlength(A): Lunghezza di A. Dà come risultato un valore decimale che rappresenta la lunghezza del vettore A. Può essere usato per calcolare la distanza tra due punti. Distanza=vlength(B-A). La formula è vlength=sqrt(vdot(A,A)).

vnormalize(A): Normalizza il vettore A. Dà come risultato un vettore di lunghezza unitaria con la stessa direzione del vettore A (*versore*). La formula è vnormalize=A/vlength(A).

vrotate(A,B): Ruota A intorno all'origine di una quantità pari a B. Assegnate le coordinate x, y, z di un punto nello spazio designato dal vettore A, la funzione ruota quel punto attorno all'origine di una quantità specificata dal vettore B. Ruota quel punto intorno all'asse x di un angolo espresso in gradi corrispondente alla prima componente di B. Allo stesso modo, B.y e B.z specificano l'ammontare in gradi della rotazione riferita all'asse y e all'asse z. Il risultato è un vettore contenente le nuove coordinate x, y, z del punto.

7.1.8.3 Funzioni Stringa

Le seguenti sono funzioni che prendono una o più stringhe e parametri decimali e danno come risultato stringhe o valori decimali. Chiamiamo S1 e S2 due espressioni valide che rappresentano due stringhe, e A, L e P espressioni valide che danno come risultato valori decimali.

asc(S1): Valore ASCII di S1. Dà come risultato un numero intero compreso nell'intervallo 0 - 255 che è il valore ASCII del primo carattere di S1. Per esempio asc("ABC") è 65 perché quello è il valore del carattere "A".

chr(A): Carattere il cui valore ASCII è A. Dà come risultato una stringa di un solo carattere. Il valore ASCII del carattere è specificato da un numero intero A compreso nell'intervallo che va da 0 a 255. Per esempio chr(70) è la stringa "F". Se usi `chr(...)` quando operi il rendering dell'oggetto testo dovresti sapere che i caratteri soggetti all'elaborazione per valori di A maggiori di 127 dipendono dal carattere TTF usato. Molti caratteri True Type usano il set di caratteri Latin-1 (ISO 8859-1), ma non tutti.

concat(S1,S2,[S3...]): Concatena le stringhe S1 e S2 ed eventuali stringhe successive. Dà come risultato una stringa che è la concatenazione di tutti i parametri stringa. Devi avere almeno 2 parametri ma chiaramente ne puoi avere di più. Per esempio:

```
concat("Il valore è ", str(A,3,1), " pollici")
```

Se il valore decimale A era 12,34 il risultato è : "Il valore è 12,3 pollici" che rappresenta una stringa.

file_exists(S1): Ricerca il file specificato da S1. Tenta di aprire il file il cui nome è specificato dalla stringa S1. La ricerca viene operata sulla directory corrente e in tutte quelle specificate in ogni opzione Library_Path .INI o attraverso il parametro +L della linea di comando. Il file è chiuso immediatamente. Dà come risultato il valore booleano pari ad 1 in caso di successo, a 0 in caso di un fallimento nella ricerca.

str(A,L,P): Converte il valore decimale A in una stringa. Dà come risultato una stringa che rappresenta il valore decimale A. Il parametro decimale L specifica la lunghezza minima della stringa ed il tipo di riempimento dello spazio a sinistra usato se la rappresentazione della stringa è più corta del minimo. Se L è positivo allora l'area a sinistra è riempita da spazi vuoti. Se L è negativo allora l'area a sinistra è riempita con zeri. L'intera lunghezza minima della stringa è data da `abs(L)`. Se la stringa necessita di una lunghezza maggiore , essa sarà allungata quanto necessario per rappresentare il valore.
Il parametro decimale P specifica il numero di cifre dopo il punto decimale. Se P è negativo allora viene usata una precisione di default come specificato dal compilatore. Qui di seguito alcuni esempi:

```
str(123.456,0,3) "123.456"
str(123.456,4,3) "123.456"
str(123.456,9,3) " 123.456"
str(123.456,-9,3) "00123.456"
str(123.456,0,2) "123.46"
str(123.456,0,0) "123"
str(123.456,5,0) " 123"
str(123.000,7,2) " 123.00"
str(123.456,0,-1) "123.456000" (secondo specifica di default)
```

strcmp(S1,S2): Opera il confronto della stringa S1 con S2. Dà come risultato un valore pari a zero se le stringhe sono uguali, un numero positivo se S1 viene dopo S2 nella sequenza ASCII, negativo in caso contrario.

strlen(S1): Lunghezza di S1. Dà come risultato un valore intero che rappresenta il numero di caratteri della stringa S1.

strlwr(S1): Lettere minuscole di S1. Dà come risultato una nuova stringa nella quale tutte le lettere della stringa S1 sono convertite in minuscolo. La stringa originale non è tagliata. Per esempio `strlwr("Saluti a Tutti!")` dà come risultato "saluti a tutti! ".

substr(S1,P,L): Sotto stringa di S1. Dà come risultato una stringa che è un sottoinsieme di caratteri della stringa S1 partendo dalla posizione specificata dal valore intero P, per una lunghezza specificata dal valore intero L. Per esempio `substr("ABCDEFGHI",4,2)` porta alla stringa "EF." Se P+L è maggiore di strlen(S1) allora si verifica un errore.

strupr(S1): Lettere maiuscole di S1. Dà come risultato una nuova stringa nella quale tutte le lettere minuscole nella stringa S1 sono convertite in lettere maiuscole. La sequenza originale non è

tagliata. Per esempio `strupr("Saluti a Tutti!")` da' come risultato "SALUTI A TUTTI! ".

val(S1): Converte la stringa S1 a un valore decimale. Da come risultato un valore decimale che è rappresentato dal testo in S1. Per esempio `val("123.45")` è 123.45 come valore decimale.

7.2 Istruzioni del Linguaggio

Il linguaggio di descrizione delle scene in POV-Ray contiene molte frasi chiamate *istruzioni del linguaggio* che indicano all'elaboratore del file come svolgere il suo lavoro. Queste istruzioni possono apparire quasi ovunque all'interno del file di scena, anche al centro di alcune istruzioni. Sono usate per includere altri file di testo nell'insieme di comandi, dichiarare identificatori, definire istruzioni condizionali o cicliche e per controllare altri importanti aspetti nell'elaborazione del file di scena.
Ciascuna istruzione comincia col carattere # . Esso è seguito da una parola chiave e facoltativamente da altri parametri.
Nelle versioni precedenti di POV-Ray, l'uso del carattere # era opzionale. Le istruzioni del linguaggio potevano essere usate solo tra oggetti o frasi riguardanti la macchina fotografica e le sorgenti luminose e non potevano apparire all'interno di queste. L'eccezione era `#include` che poteva essere presente ovunque. Ora che tutte le istruzioni del linguaggio possono essere usate quasi dovunque il carattere # è obbligatorio.
Le seguenti parole chiave precedono istruzioni del linguaggio.

```
#break
#case
#debug
#declare
#default
#else
#end
#render
#statistics
#switch
#version
#warning
```

Versioni precedenti di POV-Ray consideravano #max_intersections e #max_trace_level come delle istruzioni del linguaggio, ora sono invece parte delle impostazioni generali. (*global_settings*). E' ancora consentito il loro uso come istruzione, ma questo genera un avvertimento e potrebbe essere escluso nel futuro.

7.2.1 File Include

Il linguaggio della scena consente di specificare i file *include* ponendo la linea:

```
#include "nome_del_file.inc"
```

in ogni punto del file di input. Il `nome_del_file` può essere specificato usando ogni stringa valida, ma è comunemente rappresentato da una stringa di lettere inclusa tra virgolette. Può essere lunga fino a 40 caratteri (o i limiti del tuo computer), inclusi i due caratteri delle virgolette.
Il file include viene letto come se fosse inserito per intero in quel punto del file di scena. Usando `#include` è come se tagliassi e copiassi l'intero contenuto di questo file all'interno della tua scena. I file include possono definiti anche all'interno di altri file dello stesso tipo (possono essere annidati

l'uno nell'altro). Puoi avere al massimo 10 file include annidati. Non c'è nessuno limite per i file include non annidati.
Generalmente, i file include contengono dati per le scene ma non sono scene vere e proprie. Per convenzione i file di scena hanno estensione **.pov** ed i file include hanno estensione **.inc**.
E' consentito specificare informazioni contenute in altri drive e directory nelle indicazioni del file di scena, comunque lo si dovrebbe evitare perché ciò rende i file di scena meno facilmente trasportabili tra computer diversi.
E' comune mettere i file include 'standard' in una speciale sotto-directory. POV-Ray può leggere i file solo nella directory corrente o in una specificata dall'opzione `Library_Path` (Vedere sezione "Percorsi delle Librerie").

7.2.2 Dichiarazioni

Gli identificatori possono essere dichiarati e richiamati successivamente per rendere il file di scena più leggibile e per parametrizzare le scene così che cambiando una singola dichiarazione sia possibile cambiare molti valori. Ci sono molti identificatori incorporati che POV-Ray dichiara per te. Vedi "Identificatori Incorporati" per maggiori dettagli.

7.2.2.1 Identificatori di Dichiarazione

Un identificatore è dichiarato come segue.

```
#declare IDENTIFICATORE = OGGETTO
```

Dove IDENTIFICATORE è il nome dell' identificatore lungo fino a 40 caratteri ed OGGETTO è uno dei seguenti:

```
espressione decimale, vettoriale, di colore o di stringa
oggetti (tutti i generi)
texture, pigment, normal, finish o halo
color_map, pigment_map, slope_map, normal_map
camera, light_source
atmosphere
fog
rainbow
sky_sphere
trasformazione
```

Di seguito alcuni esempi.

```
#declare Rows = 5
#declare Count = Count+1
#declare Here = <1,2,3>
#declare White = rgb <1,1,1>
#declare Cyan = color blue 1.0 green 1.0
#declare Font_Name = "ariel.ttf"
#declare Ring = torus {5,1}
#declare Checks = pigment { checker White, Cyan }

object{ Rod scale y*5 } // non "cylinder { Rod }"
object {
Ring
pigment { Checks scale 0.5 }
```

```
transform Skew
}
```

Le dichiarazioni, come la maggior parte delle istruzioni del linguaggio, possono apparire ovunque nel file, anche al centro di altre frasi. Per esempio:

```
#declare Qui=<1,2,3>
#declare Conta=0 // inizio Conta

union {
object { Rod translate Here*Count }
#declare Count=Count+1 // ridichiarazione dentro union

object { Rod translate Here*Count }
#declare Count=Count+1 // ridichiarazione dentro union
object { Rod translate Here*Count }
}
```

Come questo esempio mostra, puoi ri-dichiarare un identificatore e puoi usare valori già dichiarati in quella
ri-dichiarazione. Comunque se tenti di ri-dichiarare un identificatore diverso da ciò a cui fa riferimento, verrà generato un messaggio di avvertimento.
Le dichiarazioni possono essere annidate una dentro l'altra senza limiti. Nell'esempio della sezione precedente potresti dichiarare l'intera unione come un oggetto. Comunque per ragioni tecniche non puoi usare alcuna istruzione del linguaggio all'interno di dichiarazioni che facciano riferimento a valori decimali, vettoriali o espressioni del colore.

7.2.3 Istruzione Predefinita

POV-Ray crea una texture predefinita quando inizia l'elaborazione. Puoi cambiare quelle predefinite come descritto di seguito. Ogni volta che specifichi una dichiarazione di `texture{...}`, POV-Ray crea una copia della texture predefinita. Qualunque cosa venga inserita nella dichiarazione di texture avrà la precedenza sulle impostazioni predefinite. Se inserisci un pigmento, una normale o una finitura ad un oggetto senza nessuna dichiarazione di texture allora POV-Ray verifica se una texture sia già stata specificata. Se l'oggetto ha una texture, allora il pigmento, la normale o la finitura modificherà la texture esistente. Se nessuna texture è stata specificata per quell'oggetto allora la texture predefinita viene copiata ed il pigmento, la normale o la finitura modificherà quella texture.
Puoi cambiare la texture predefinita, il pigmento, la normale o la finitura usando l'istruzione `#default{...}` come di seguito:

```
#default {
texture {
pigment {...}
normal {...}
finish {...}
}
}
```

O puoi cambiarne una parte in questo modo:

```
#default {
```

```
pigment {..}.
}
```

Ciò consente comunque di cambiare il pigmento della texture predefinita. Può essere definita solo una texture predefinita per volta, formata dal pigmento, dalla normale e dalla finitura. L'esempio considerato prima non crea un pigmento separato predefinito. Nota che le texture speciali come 'checker' o le mappature di materiali (`material_map`) o di texture (`texture_map`) non possono essere usati come elementi predefiniti.

Puoi cambiare le dichiarazioni predefinite molte volte all'interno di una scena. Le successive dichiarazioni `#default` iniziano da quelle che avevano effetto fino a quel momento. Se ad un certo punto desideri riprendere le dichiarazioni predefinite di POV-Ray, dovresti prima salvarle in questo modo:

```
// All'inizio del file
#declare Original_Default = texture {}
```

successivamente dopo aver cambiato le impostazioni predefinite puoi ripristinarle con...

```
#default {texture {Original_Default}}
```

Se non specifichi una texture per un oggetto allora la texture predefinita verrà assegnata ad esso quando compare nella scena. Non verrà assegnata quando l'oggetto è solo dichiarato. Per esempio:

```
#declare My_Object =
sphere{ <0,0,0>, 1 } // non c'è texture predefinita
object { My_Object } // la texture predefinita viene assegnata qui
```

Puoi forzare POV-Ray ad utilizzare una texture predefinita aggiungendo una dichiarazione di texture vuota come segue:

```
#declare My_Thing =
sphere { <0,0,0>, 1 texture {} } // Texture predefinita applicata
```

Le condizioni originali predefinite di POV-Ray per ogni voce sono fornite nella documentazione in ogni appropriato paragrafo.

7.2.4 Istruzioni di Versione

Sebbene siano stati introdotti molti cambiamenti nel linguaggio di descrizione della scena in POV-Ray 3.0, tutta la sintassi relativa alla versione 2.0 e la maggior parte di quella della versione 1.0, funzionano ancora. Ogni qual volta sia possibile cerchiamo di mantenere la compatibilità con le versioni precedenti. Una caratteristica introdotta nella versione 2.0, che era incompatibile con ogni file di scena della versione 1.0 è l'elaborazione delle espressioni decimali. Utilizzando l'impostazione mediante il parametro della linea di comando `+ MV1.0` o con l'opzione `Version=1.0` nel file .INI si disattivano l'analisi delle espressioni decimali e molti messaggi di avvertimento e quasi tutti i file della versione `1.0` potranno essere elaborati ancora. I cambiamenti tra la versione `2.0` e `3.0` non sono stati così radicali. L'impostazione `Version=2.0` è necessaria solo per eliminare alcuni messaggi di avvertimento. Naturalmente l'impostazione predefinita per questa opzione è `Version=3.0`.

L'istruzione di linguaggio `#version` è usata per cambiare modalità all'interno dei file di scena,

mentre il parametro della linea di comando o l'opzione del file .INI incide solo sull'impostazione iniziale.

Insieme con l'identificatore incorporato `version`, l'istruzione `#version` consente di salvare e ripristinare i precedenti valori di quest'impostazione relativi alla compatibilità. Per esempio supponi che il file **mystuff.inc** sia nel formato della versione 1.0. All'inizio del file si potrebbe porre:

```
#declare Temp_Vers = version // Salva il valore precedente
#version 1.0 // Cambia nella modalità 1.0

... // La versione 1.0 inizia qui ...

#version Temp_Vers // Ripristina la versione precedente
```

Versioni precedenti di POV-Ray non ti avrebbero consentito di cambiare versione all'interno di un oggetto o di una dichiarazione ma questa restrizione è stata eliminata in POV-Ray 3.0.

Le future versioni di POV-Ray potrebbero continuare a non avere una piena compatibilità con le sintassi precedenti nonostante l'istruzione `#version`. Cercheremo per quanto possibile di rimanere in linea con la sintassi della versione 3.0.

7.2.5 Istruzioni Condizionali

POV-Ray 3.0 consente una varietà di nuove istruzioni del linguaggio per implementare analisi condizionali delle varie sezioni del tuo file di scena. Questo è particolarmente utile per descrivere il moto degli oggetti per le animazioni, ma ha anche altri usi. E' anche utile in una istruzione di ciclo `#while`. E' possibile annidare le istruzioni condizionali fino a una profondità di 200 livelli.

7.2.5.1 Istruzioni IF ELSE

L'Istruzione condizionale più semplice è quella tradizionale `#if` (se). Essa si esprime nella forma...

```
#if (COND)
// Questa sezione è
// analizzata se COND è vera
# else
// Questa sezione è
// analizzata se COND è falsa
#end // Fine della parte condizionale
```

dove (COND) è un'espressione decimale che riporta a un valore booleano. Un valore di 0.0 è falso ed ogni valore diverso da zero è vero. Nota che valori estremamente bassi come circa 1e-10 sono considerati uguali a zero (e quindi falsi). Sono richieste le parentesi intorno alla condizione. L'istruzione `#else` è opzionale. L'istruzione `#end` è necessaria.

7.2.5.2 Istruzioni IFDEF

L'istruzione `#ifdef` è simile all'istruzione `#if`; è comunque usata per determinare se un identificatore è stato dichiarato in precedenza. Dopo l'istruzione `#ifdef` invece di un'espressione booleana si pone un solo identificatore chiuso tra parentesi. Per esempio:

```
#ifdef (User_Thing)
// Questa sezione è analizzata se
```

```
// l'identificatore "User_Thing" era
// precedentemente dichiarato
object{User_Thing} // invoca identificatore
# else
// Questa sezione è analizzata se
// l'identificatore "User_Thing" non era
// precedentemente dichiarato
box{<0,0,0>,<1,1,1>} // usa un valore preimpostato
# end
// Fine della parte condizionale
```

L'istruzione `#else` è opzionale. L'istruzione `#end` è necessaria.

7.2.5.3 Istruzioni IFNDEF

L'istruzione `#ifndef` è simile all'istruzione `#ifdef`, comunque è usata per determinare se l'identificatore dato non è ancora dichiarato. Per esempio:

```
#ifndef (User_Thing)
// Questa sezione è analizzata se
// l'identificatore "User_Thing" non era stato
// precedentemente dichiarato
box{<0,0,0>,<1,1,1>} // usa un valore preimpostato
# else
// Questa sezione è analizzata se
// l'identificatore "User_Thing" era stato
// precedentemente dichiarato
object{User_Thing} //invoca identificatore
# end
// Fine della parte condizionale
```

L'istruzione `#else` è opzionale. L'istruzione `#end` è necessaria.

7.2.5.4 Istruzioni SWITCH CASE e RANGE

Un'istruzione condizionale piu potente è `#switch`. La sintassi è la seguente...

```
#switch (VALORE)
#case (TEST_1)
// Questa sezione è analizzata se VALORE=TEST_1
#break // Fine primo caso

#case (TEST_2)
// Questa sezione è analizzata se VALORE=TEST_2
#break // Fine secondo caso

#range (LOW_1, HIGH_1)
// Questa sezione è analizzata se (VALORE>=LOW_1)&
(VALORE<=HIGH_1)
#break // Fine terzo caso

#range(LOW_2, HIGH_2)
// Questa sezione è analizzata se (VALORE>=LOW_2)&
```

```
(VALORE<=HIGH_2)
#break // Fine quarto caso

#else
// Questa sezione è analizzata se nessun altro case o
// range sono veri.
#end // Fine della parte condizionale
```

L'espressione decimale VALORE che segue l'istruzione `#switch` è valutata e comparata ai valori nelle istruzioni `#case` o `#range`. Quando si usa `#case`, essa è seguita da un'espressione decimale (TEST_1 nell'esempio) in parentesi. Essa è comparata a VALORE. Come al solito in POV-Ray, i confronti di valori decimali sono considerati uguali se la loro differenza è inferiore a 1e-10. Se i valori sono uguali, l'analisi continua normalmente fino ad incontrare l'istruzione `#break`, `#else` o `#end`. Se il confronto fallisce POV-Ray salta le istruzioni fino a quando non trova `#case` o `#range`.

Se usi l'istruzione `#range` essa è seguita da due espressioni decimali (LOW_1 ed HIGH_1 nell'esempio) che sono chiuse in parentesi e separate da una virgola. Se l'interruttore (switch) VALORE cade nell'intervallo specificato allora l'analisi continua normalmente fino a che non è raggiunta l'istruzione `#break`, `#else` o `#end`. Se VALORE è al di fuori dall'intervallo il confronto fallisce e POV-Ray salta fino a quando non viene trovato un altro `#case` o `#range`.

Se nessuna istruzione `#case` o `#range` ha successo è analizzata la sezione `#else`. L'istruzione `#else` è opzionale. Se non è specificata nessuna istruzione `#else` e nessun confronto riesce, allora l'analisi riprende dopo l'istruzione `#end`.

Ci può essere qualsiasi numero di istruzioni `#case` o `#range` in qualsiasi ordine si voglia. Se una sezione viene valutata vera ma non è specificato alcuna istruzione `#break`, l'analisi fallirà al successivo `#case` o `#range` e continuerà fino a un `#break`, `#else` o `#end`. Incontrando un'istruzione `#break` mentre il programma è in fase di elaborazione, una sezione che ha successo provoca un immediato salto all'istruzione `#end` senza operare l'analisi delle successive sezioni, anche se una condizione successiva poteva essere soddisfatta.

7.2.5.5 Istruzione WHILE

L'istruzione `#while` consente operazioni cicliche che rendono semplice il disporre oggetti multipli in un modello o per altri usi. L'istruzione `#while` è seguita da un'espressione decimale che riporta a un valore booleano. Un valore pari a 0.0 è falso ed ogni valore diverso da zero è vero. Nota che valori estremamente piccoli vicini a 1e-10 sono considerati zero. Le parentesi intorno all'espressione sono necessarie. Se la condizione è vera l'analisi continua normalmente finché non si incontra un'istruzione `#end`. Raggiunta `#end`, POV-Ray ritorna indietro all'istruzione `#while` e la condizione viene elaborata nuovamente. Il ciclo continua fino a che la condizione fallisce. Quando fallisce, l'elaborazione continua da dopo l'istruzione `#end`. Per esempio:

```
#declare Conteggio = 0
#while (Conteggio < 5)
object {Mio_Oggetto translate x*3*Conteggio}
#declare Conteggio = Conteggio+1
#end
```

Questo esempio pone cinque copie di Mio_Oggetto in una fila, distanziate di tre unità nella direzione x.

7.2.6 Istruzioni Messaggio Utente

Con l'aggiunta delle istruzioni condizionali e cicliche, il linguaggio di POV-Ray ha la possibilità di essere come un linguaggio di programmazione attuale. Questo vuol dire che è necessario avere alcuni strumenti per verificare come il programma sta procedendo quando si cerca di correggere errori di programmazione relativi a istruzioni cicliche e condizionali. Per far questo occorre abilitarlo a mostrare sullo schermo i messaggi. Hai cinque possibili scelte per abilitare il programma ad avvertirti di possibili errori se ciò ti può essere utile. Per questo metodo è disponibile una configurazione limitata a produrre dei messaggi.

7.2.6.1 Flusso di Messaggi Testo

La sintassi per una comunicazione di testo è una delle seguenti:

```
#debug STRINGA
#error STRINGA
#error STRINGA
#render STRINGA
#statistics STRINGA
#warning STRINGA
```

Dove STRINGA è ogni sequenza valida di testo, inclusi identificatori o funzioni stringa che riportano una sequenza di caratteri. Per esempio:

```
#switch (clock*360)
#range (0,180)
#render "Clock da 0 a 180 \n"
#break
#range (180,360)
#render "Clock da 180 a 360 \n"
#break
#else
#warning "Clock al di fuori dell'intervallo \n"
#warning concat("il valore è :",str(clock*360,5,0),"\n")
#end
```

Ci sono sette flussi distinti di testo che POV-Ray usa per comunicare con l'utente. Puoi produrre output solo su cinque. Su alcune versioni di POV-Ray, ciascun flusso è indicato da un colore particolare. Il testo di questi flussi è mostrato ogni qualvolta è necessario, quindi spesso è una miscela dei vari tipi. La distinzione è importante solo se scegli di disabilitare alcuni dei flussi o indirizzare alcuni dei flussi su file di testo. Su alcuni sistemi sei in grado di fare una rassegna dei flussi separatamente nel loro scorrere. Vedi " Output di Testo su Console" (6.2.5.2) per dettagli sul ri-indirizzare i flussi di output su un file di testo.

Qui vi è una descrizione di come POV-Ray usa ciascun flusso. Puoi usarli per qualunque scopo tu voglia, notando che l'uso dell'output `#error` causa un errore 'fatale' dopo che viene mostrato il testo (e termina il programma).

DEBUG: Questo flusso mostra messaggi per la correzione. E' progettato principalmente per i programmatori, ma insieme ad altri flussi può essere anche usato dagli utenti per mostrare messaggi dall'interno dei loro file di scena.

FATAL: Questo flusso mostra messaggi di errore inevitabile. Dopo che è comparso questo messaggio, POV-Ray termina. Quando l'errore è un errore dell'analisi della scena, ti possono essere mostrate alcune linee di testo della scena che conducono fino all'errore.

RENDER: Questo flusso mostra l'informazione relativa a quali opzioni hai specificato per operare il rendering della scena. Include una panoramica su tutte le principali opzioni come il nome della scena, la risoluzione, le impostazioni dell'animazione, l'anti-aliasing ed altri.

STATISTICS: Questo flusso mostra informazioni statistiche dopo che è stato operato il rendering. Include informazioni relative al numero di raggi tracciati, il tempo trascorso per l'operazione di rendering ed altre informazioni.

WARNING: Questo flusso mostra i messaggi di avvertimento durante l'analisi del file di scena ed altri avvertimenti. Nonostante l'avvertimento, POV-Ray può continuare ad operare il rendering della scena.

I flussi **BANNER** e **STATUS** non sono accessibili all'utente.

7.2.6.2 Formattazione del Testo

Alcune sequenze di testo sono disponibili per includere *caratteri di controllo* non stampabili nel testo. Queste sequenze sono simili a quelle usate nelle costanti stringa nel linguaggio di programmazione C. Nota che questi caratteri di controllo si applicano solo nelle istruzioni di messaggio. Non sono implementate per altro uso in POV-Ray, come ad esempio per definire oggetti testo o nomi di file. In base alla piattaforma usata, possono non essere completamente supportati per messaggi console. Comunque essi appariranno in qualsiasi file di testo se indirizzi un flusso di testo su file.
Le sequenze sono:

"\a"	Beep o allarme, 0x07
"\b"	Ritorno (Backspace), 0x08
"\f"	Avanzamento carta, 0x0C
"\n"	Linea nuova (avanzamento di una interlinea) 0x0A
"\r"	Ritorno del carrello 0x0D
"\t"	Tabulazione orizzontale 0x09
"\v"	Tabulazione verticale 0x0B
"\0"	Nullo 0x00
"\\"	Backslash 0x5C
"\'"	Apice singolo 0x27
"\""	Doppio apice 0x22

Per esempio:

```
# debug "Questa è una linea.\nMa questa è un\'altra"
```

7.3 Il Sistema di Coordinate di POV-Ray

Gli oggetti, le luci e il punto di osservazione sono posizionati usando un tipico sistema di coordinate 3D. Il sistema di coordinate usato normalmente da POV-Ray ha l'asse positivo delle y rivolto verso l'alto, l'asse positivo delle x rivolto a destra e l'asse positivo delle z che punta in avanti verso lo schermo. Le direzioni negative degli assi sono rivolte in direzione contraria a quella appena esposta,

come mostrato nell'immagine della sezione "Capire il sistema di coordinate di POV-Ray".

La posizione di un punto in questo sistema di coordinate è normalmente definita da un vettore a tre componenti. I tre valori corrispondono rispettivamente alle direzioni degli assi x, y e z . Per esempio, il vettore < 1, 2, 3 > indica il punto che si trova un'unità a destra, due unità verso l'alto e tre unità di fronte rispetto al centro dell'universo che si trova per convenzione a < 0, 0, 0 >.

I vettori comunque non rappresentano sempre punti. Essi si possono riferire a un valore di ridimensionamento, a una traslazione o a una rotazione di un elemento della scena o per modificare la texture applicata a un oggetto.

Le trasformazioni consentite sono la rotazione, il ridimensionamento (*scaling*) e la traslazione. Esse sono usate per ruotare, dimensionare o traslare un oggetto o una texture. Può essere usata anche una trasformazione *matrice* per specificare direttamente trasformazioni complesse.

7.3.1 Trasformazioni

Le trasformazioni consentite sono la rotazione, il ridimensionamento e la traslazione. Esse sono usate per ruotare, dimensionare o traslare un oggetto o una texture.

```
rotate <VETTORE>
scale <VETTORE>
translate <VETTORE>
```

7.3.1.1 Traslazioni

Un oggetto o una texture si possono spostare aggiungendo il parametro di traslazione. Esso consiste della parola chiave `translate` seguita da un'espressione vettoriale. I termini del vettore specificano il numero di unità di cui spostare l'oggetto in ognuna delle direzioni x, y, e z. La traslazione muove l'elemento a cui si riferisce dalla sua posizione corrente. Per esempio

```
sphere { <10, 10, 10>, 1
pigment { Green }
translate <-5, 2, 1>
}
```

muove la sfera da < 10, 10, 10 > a < 5, 12, 11>. Questa operazione non muove l'oggetto nella posizione assoluta < -5, 2, 1 >. Una traslazione di zero in una delle tre direzioni lascerà l'elemento immutato rispetto a quell'asse. Per esempio:

```
sphere { <10, 10, 10>, 1
pigment { Green }
translate 3*x // viene considerato come < 3, 0, 0 > così muove di
3 unità
// nella direzione x e lascia inalterato lungo le y e le z
}
```

7.3.1.2 Ridimensionamento

Puoi cambiare la dimensione di un oggetto o di una texture aggiungendo il parametro di ridimensionamento. Esso consiste della parola chiave `scale` seguita da un'espressione vettoriale. I 3 termini del vettore specificano l'ammontare del ridimensionamento in ognuna delle direzioni x, y e z.

Il ridimensionamento è usato per allungare o schiacciare un elemento della scena. Valori più grandi di uno allungano l'elemento sull'asse a cui fa riferimento lo scaling, mentre valori minori dell'unità sono usati per schiacciarlo lungo quell'asse. Il ridimensionamento è relativo alla dimensione *attuale* dell'elemento. Se l'elemento è stato precedentemente ridimensionato usando il comando `scale` allora il cambio di scala sarà relativo alla nuova dimensione. Ciò significa che questo comando non opera sulla dimensione che l'oggetto poteva avere originariamente. Possono essere usati valori multipli di ridimensionamento.
Per esempio

```
sphere { <0,0,0>, 1
scale <2,1,0.5>
}
```

allungherà e plasmerà la sfera in una forma ellissoidale che è lunga il doppio rispetto all'originale lungo l'asse x, uguale sull'asse y e dimezzata, sempre rispetto alla dimensione originale nella direzione z.
Se è specificato un solo valore decimale, esso verrà riferito a tutte e tre le componenti vettoriali. Così l'oggetto è scalato uniformemente dello stesso valore in tutte le direzioni. per esempio:

```
object {
MioOggetto
scale 5 // Viene considerato come <5, 5, 5 > quindi lo scaling
sarà uniforme
// in ogni direzione.
}
```

7.3.1.3 Rotazioni

Puoi cambiare l'orientamento di un oggetto o di una texture aggiungendo un parametro di rotazione. Esso consiste della parola chiave `rotate` seguita da un'espressione vettoriale. I tre termini del vettore specificano la rotazione in gradi relativa ad ognuno degli assi x, y e z *attorno all'origine*.

Fai attenzione al fatto che l'ordine delle rotazioni è importante. Le rotazioni devono riferirsi prima all'asse x, poi all'asse y e infine all'asse z. Se non sei sicuro che questo è ciò che vuoi, dovresti operare la rotazione intorno ad un asse alla volta, usando rotazioni multiple per ottenere la giusta posizione finale. Come in

```
rotate <0, 30, 0> // 30 gradi intorno all'asse Y poi,
rotate <-20, 0, 0> // -20 gradi intorno all'asse X poi,
rotate <0, 0, 10> // 10 gradi intorno all'asse Z.
```

La rotazione è sempre intorno ad un asse. Così, se un oggetto è a una certa distanza dall'asse di rotazione, esso non solo ruoterà ma *orbiterà* intorno all'asse stesso modificando la sua posizione nello spazio.

Per comprendere le istruzioni sulla rotazione degli oggetti dovresti eseguire il famoso esercizio di 'Aerobica per Grafica Computerizzata', come spiegato nella sezione "Capire il sistema di coordinate di POV-Ray".

7.3.1.4 La Parola Chiave MATRIX

La parola chiave `matrix` può essere usata per specificare esplicitamente la matrice di trasformazione da applicare ad oggetti o a texture. La sua sintassi è:

```
matrix < m00, m01, m02,
m10, m11, m12,
m20, m21, m22,
m30, m31, m32 >
```

Dove gli elementi da m00 a m32 sono termini (decimali) di una matrice di trasformazione quadrata 4*4 con la quarta colonna implicitamente impostata a < 0, 0, 0, 1 >. Un punto P di coordinate P=< px, py, pz >, è trasformato nel punto Q, di coordinate Q=< qx, qy, qz > dalle trasformazioni:

```
qx = M00 * px + M10 * py + M20 * pz + M30
qy = M01 * px + M11 * py + M21 * pz + M31
qz = M02 * px + M12 * py + M22 * pz + M32
```

Normalmente, non si userà la parola chiave matrix in quanto essa è di utilizzo meno immediato rispetto ai semplici comandi di trasformazione e più difficile da visualizzare. C'è però un interessante aspetto del comando `matrix`. Esso consente trasformazioni più generali come lo schiacciamento lungo un piano. La matrice seguente porta a deformare un oggetto lungo l'asse y.

```
object {
MioOggetto
matrix < 1, 1, 0,
0, 1, 0,
0, 0, 1,
0, 0, 0 >
}
```

7.3.2 Ordine delle Trasformazioni

Poiché le rotazioni sono sempre relative agli assi e il ridimensionamento è relativo all'origine del sistema di riferimento, in generale si creano gli oggetti nell'origine, poi si scalano e ruotano. Infine si traslano nella posizione opportuna. E' un errore comune quello di posizionare correttamente un oggetto e poi decidere di ruotarlo; ciò comporterà una rotazione dell'oggetto *intorno* ad un asse e la sua posizione potrebbe cambiare a tal punto da farlo uscire fuori dal campo visivo!

Allo stesso modo, ridimensionare un oggetto dopo una traslazione può comportare uno spostamento inaspettato dell'oggetto medesimo. Se operi il ridimensionamento prima della traslazione il cambio di scala moltiplicherà il valore della traslazione stessa. Per esempio:

```
translate <5, 6, 7>
scale 4
```

Traslerà l'oggetto della trasformazione a < 20, 24, 28 > invece che a < 5, 6, 7 >. Poni attenzione all'ordine corretto quando operi le trasformazioni.

7.3.3 Identificatori di Trasformazione

A volte è utile combinare insieme molte trasformazioni ed applicarle in diverse circostanze. Un identificatore di trasformazione può essere usato a tale scopo. Gli identificatori di trasformazione sono dichiarati nel modo seguente:

```
#declare IDENTIFICATORE = transform { TRASFORMAZIONE... }
```

Dove IDENTIFICATORE è l'identificatore dichiarato e TRASFORMAZIONE è la combinazione di una o più traslazioni, rotazioni, ridimensionamenti, specificazioni `matrix`, o anche un identificatore di trasformazione dichiarato precedentemente. Un identificatore di trasformazione è richiamato dalla parola chiave `transform` senza alcuna parentesi come mostrato di seguito:

```
object {
MioOggetto // Produci una copia di MioOggetto
transform MiaTrasformazione // Applica la trasformazione
translate -x*5 // Poi muovi di 5 unità a sinistra
}
object {
MioOggetto // Produci un'altra copia di MioOggetto
transform MiaTrasformazione // Applica la stessa trasformazione
translate x*5 // Poi muovi di 5 unità a destra
}
```

Su oggetti CSG (di Geometria Solida Costruttiva) estremamente complessi, se applichi trasformazioni dichiarate invece di singole trasformazioni, rotazioni, ridimensionamenti o istruzioni matriciali puoi accelerare l'elaborazione del file della scena. La trasformazione è applicata *una* volta a ciascun componente. In generale applicare ogni singola traslazione, rotazione, cambio di dimensione o istruzioni matriciali, prende molto tempo. Questa procedura accorcia i tempi di elaborazione e di rendering in entrambe le situazioni.

7.3.4 Trasformare le Texture e gli Oggetti

Quando un oggetto subisce una trasformazione, tutte le texture ad esso applicate vengono modificate di conseguenza. Ciò significa che se tu hai operato una traslazione, una rotazione, un ridimensionamento o una trasformazione matriciale prima di avere applicato una texture, questa non verrà modificata. Se la trasformazione è stata operata dopo l'applicazione della texture, allora quest'ultima seguirà le modifiche dell'oggetto. Se la trasformazione è interna all'istruzione di texture (`texture {...}`) allora solo la texture ne sarà soggetta. La forma rimane la stessa. Per esempio:

```
sphere { 0, 1
texture { Jade } // identificatore di texture da TEXTURES.INC
scale 3 //questo ridimensionamento è applicato
// sia alla sfera che alla texture
}

sphere { 0, 1
scale 3 // questo ridimensionamento è applicato solo alla sfera
texture { Jade }
}

sphere { 0, 1
texture {
Jade
scale 3 // Questo ridimensionamento è applicato solo alla texture
}
}
```

Le trasformazioni possono essere anche applicate indipendentemente a un pattern di pigmenti o di normali. Nota che il ridimensionare un pattern di normali influisce solo sull'ampiezza e sulla

spaziatura. Non vengono ridimensionate l'altezza apparente o la profondità dei rilievi (*Bumps*). Per esempio:

```
box { <0, 0, 0>, <1, 1, 1>
texture {
pigment {
checker Red, White
scale 0.25 // Questo influenza solo il colore del pattern
}
normal {
bumps 0.3 // Questo specifica l'altezza apparente dei rilievi
scale 0.2 // ridimensiona il diametro e lo spazio tra i rilievi
// ma non l'altezza. Non ha effetto sul colore di pattern.
}
rotate y*45 // Questo influenza l'intera texture ma
} // non l'oggetto.
}
```

7.4 Macchina Fotografica

La definizione del punto di osservazione (macchina fotografica o *camera*), descrive la posizione, il tipo di proiezione e le sue caratteristiche nella scena. La sua sintassi è:

```
camera {
[ perspective | orthographic | fisheye |
ultra_wide_angle | omnimax | panoramic |
cylinder VALORE DECIMALE ]
location <VETTORE>
look_at < VETTORE >
right < VETTORE >
up < VETTORE >
direction < VETTORE >
sky < VETTORE >
right < VETTORE >
angle VALORE DECIMALE
blur_samples VALORE DECIMALE
aperture VALORE DECIMALE
focal_point < VETTORE >
normal { NORMALI }
}
```

In funzione del tipo di proiezione sono richiesti dei parametri, alcuni dei quali sono opzionali. Se non viene assegnato nessun tipo di proiezione viene usata la proiezione prospettica (*pinhole camera*). Se non viene assegnata alcuna camera ne viene usata una con parametri predefiniti.

A parte il tipo di proiezione, tutte le camere usano le seguenti parole chiave: `location`, `look_at`, `right`, `up`, `direction` e `sky`, per determinare l'ubicazione e l'orientamento del punto di osservazione. Il loro significato differisce con il tipo di proiezione usato. Un chiarimento più particolareggiato sulla collocazione della camera segue più tardi.

7.4.1 Tipo di Proiezione

L'elenco seguente spiega i diversi tipi di proiezione che possono essere usati con la macchina fotografica. I tipi più comuni sono la proiezione prospettica e le proiezioni ortogonali.

Proiezione prospettica: Questa proiezione simula la classica macchina fotografica, e riproduce il normale effetto di prospettiva. L'angolo (orizzontale) di osservazione è determinato dal rapporto tra la lunghezza del vettore direzione e la lunghezza del vettore `right` o dalla parola chiave opzionale `angle`, che rappresenta la scelta normalmente utilizzata. L'angolo di ampiezza dell'osservazione deve avere un'apertura compresa tra 0 e 180 gradi. Vedi la figura sotto per la geometria della macchina fotografica prospettica.

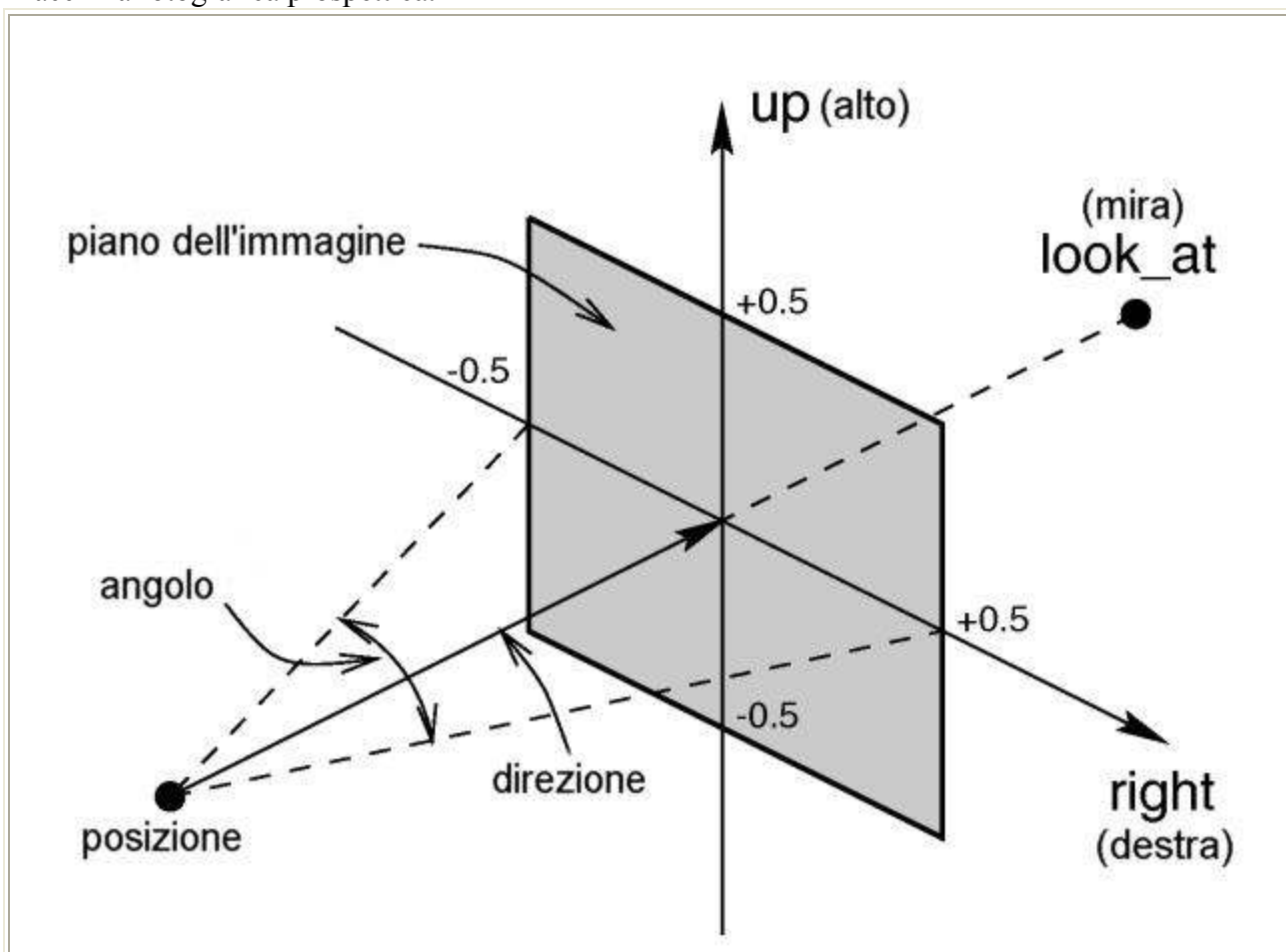



Figura 197- Proiezione Prospettica

Proiezione ortogonale: Questa proiezione usa raggi di proiezione paralleli per creare un'immagine della scena. La dimensione dell'immagine è determinata dalle lunghezze del vettore di destra e di quello rivolto verso l'alto.

Se aggiungi la parola chiave orthographic dopo tutti gli altri parametri di una macchina fotografica prospettica otterrai una vista ortogonale con la stessa area, la dimensione dell'immagine sarà cioè la stessa. In questo caso non è necessario specificare la lunghezza del vettore di destra e di quello rivolto in alto (right ed up) perché essi saranno calcolati automaticamente. Bisogna comunque essere consapevoli del fatto che le parti visibili della scena cambiano quando si commuta da prospettiva a vista ortogonale. Finché tutti gli oggetti che interessano sono vicini al punto indicato con look_at (punto di mira), essi saranno ancora visibili se usiamo la macchina fotografica ortogonale. Gli oggetti più lontani potranno invece uscire fuori dall'angolo visuale.

Proiezione Fisheye (occhio di pesce): Questa è una proiezione sferica. L'angolo dell'osservazione è

specificato dalla parola chiave angle. Un angolo di 180 gradi crea il classico fisheye mentre un angolo di 360 gradi crea un super-fisheye (visione sferica). Se usi questa proiezione dovresti ottenere un'immagine circolare, oppure ellittica. Per maggiori chiarimenti vedi "Rapporto Altezza/Larghezza".

Proiezione ad angolo ultra ampio (ultra_wide_angle): Questa proiezione è piuttosto simile al fisheye ma proietta l'immagine su un rettangolo invece che su di un cerchio. L'angolo dell'osservazione può essere specificato usando la parola chiave angle.

Proiezione Omnimax: La proiezione omnimax è un fisheye di 180 gradi che ha un angolo di osservazione ridotto nella direzione verticale. In realtà questa proiezione è usata per fare film che possono essere visti in luoghi come i teatri Omnimax. L'immagine tenderà ad essere ellittica. La parola chiave angle non è usata con questa proiezione.

Proiezione panoramica: Questa proiezione è chiamata "proiezione cilindrica equirettangolare". Supera il problema della degenerazione della proiezione prospettica quando l'angolo di osservazione si avvicina a 180 gradi. Usa un tipo di proiezione cilindrica che consente di usare angoli visuali più grandi di 180 gradi con una tollerabile distorsione laterale. La parola chiave angle è usata per determinare l'angolo di osservazione.

Proiezione cilindrica: Usando questa proiezione la scena è proiettata sopra un cilindro. Vi sono quattro diversi tipi di proiezioni cilindriche che dipendono dall'orientamento del cilindro e dalla posizione della punto di vista. L'angolo di osservazione e la lunghezza dei vettori up e right determinano le dimensioni della macchina fotografica e dell'immagine visibile. La macchina fotografica da usare è specificata da un numero. I tipi sono:

1	cilindro verticale, punto di vista fisso
2	cilindro orizzontale, punto di vista fisso
3	cilindro verticale, punto di vista che si muove lungo l'asse del cilindro
4	cilindro orizzontale, punto di vista che si muove lungo l'asse del cilindro

Se la macchina fotografica prospettica è usata con la parola chiave `angle`, ha la priorità l'angolo di osservazione specificato, se usata con la parola chiave `direction` sarà la direzione ad avere la priorità. Ogni volta che viene usato l'angolo, la lunghezza dei vettori direzione viene adattata di conseguenza.

Non vi è alcuna limitazione nell'angolo di visuale ad eccezione della proiezione prospettica. Se scegli angoli di osservazione più grandi di 360 gradi vedrai immagini ripetute della scena (il modo in cui le immagini si ripetono dipende dalla macchina fotografica). Questo potrebbe essere utile per effetti speciali.

Nota che funzione *Vista_Buffer* può essere usata solo con la macchina fotografica prospettica e con quella ortogonale.

7.4.2 Messa a Fuoco

Simula la profondità di campo inviando un numero di campioni da punti leggermente spostati all'interno di ogni pixel e calcolando la media dei risultati.

L'impostazione dell'apertura (parola chiave `aperture`) determina la profondità di campo che interessa le varie zone. Aperture grandi danno molta sfocatura, mentre aperture strette daranno una ampia zona a fuoco. Nota che, mentre in questo si comporta come una vera macchina fotografica, i

valori dell'apertura sono puramente arbitrari e non riferibili direttamente alle aperture di diaframma di una reale macchina fotografica.

Il centro della zona nitida è rappresentato dal vettore che determina il punto focale (`focal_point`) (`focal_point` ha normalmente un valore predefinito di < 0, 0, 0 >).

Il valore di `blur_samples` controlla il numero massimo di raggi utilizzati per ciascun pixel. Più raggi danno un'immagine più omogenea ma rendono il calcolo più lento, benché questa elaborazione sia controllata da un algoritmo di ottimizzazione che blocca i raggi inviati quando è stato raggiunto un certo grado di accuratezza tale che un numero maggiore di raggi non porterebbe a risultati di rilievo.

Le parole chiave `confidence` e `variance` controllano la funzione di ottimizzazione. Il valore di `confidence` è usato per determinare quando i campioni sono essere sufficientemente vicini al colore giusto. Il valore di `variance` specifica una tolleranza accettabile sulla variazione dei campioni presi fino a quel punto. In altre parole, il processo di invio dei raggi viene terminato quando il valore del colore valutato è molto probabilmente (come stimato dal valore `confidence`) vicino al vero valore del colore.

Dato che il valore di `confidence` è ottenuto da un calcolo delle probabilità, esso può variare da 0 a 1 (il valore predefinito è 0.9 ovvero il 90%). Il valore di `variance` dovrebbe trovarsi nel raggio della minima differenza di colore distinguibile (il valore predefinito è 1/128).

Un alto valore di `confidence` porterà ad un maggior numero di campioni, tempi di rendering più lunghi ed immagini migliori. Lo stesso vale per bassi valori di `variance`.

Le condizioni predefinite non prevedono l'uso della messa a fuoco, l'apertura focale è posta a 0 e il numero di campioni è 0.

7.4.3 Perturbazione dei Raggi

La parola chiave opzionale `normal` può essere usata per assegnare un pattern di normali alla macchina fotografica. Usando questo pattern tutti i raggi subiranno una perturbazione. Questo ti consente di creare effetti speciali. Vedi la scena animata **camera2.pov** come esempio.

7.4.4 Posizionare la Macchina Fotografica

Nelle seguenti sezioni ci occuperemo di come posizionare la macchina fotografica.

7.4.4.1 Posizione e Mira

In molti casi, ti saranno necessari solamente due vettori per posizionare la macchina fotografica : `location` e `look_at`. Per esempio:

```
camera{
location <3,5,-10>
look_at <0,2,1>
}
```

La parola chiave `location` indica semplicemente la posizione della macchina fotografica, specificandone le coordinate x, y, z.
La macchina fotografica può essere posizionata in ogni punto dello spazio.
La posizione di default è <0, 0, 0>.

Il vettore look_at rappresenta il punto di mira, e orienta la macchina fotografica rivolgendola verso il punto di coordinate <x, y, z>. Per default la macchina fotografica è rivolta verso un punto distante un'unità lungo l'asse z dalla sua posizione.
Il vettore look_at dovrebbe quasi sempre essere l'ultima specificazione che si inserisce nella frase camera{...}. Se si aggiungono altri parametri dopo look_at, la mira della macchina fotografica potrebbe risultarne alterata.

7.4.4.2 Il Vettore Sky

Normalmente la macchina fotografica in POV-Ray si muove a destra o a sinistra ruotando attorno all'asse y, fino a che raggiunge l'allineamento col punto stabilito dalla parola chiave look_at, quindi è inclinata verso il basso o verso l'alto fino a quando questo punto non viene perfettamente centrato. Potresti comunque desiderare che la macchina fotografica si inclini lateralmente, come un aereo in virata. E' possibile cambiare l'inclinazione della macchina fotografica usando il vettore sky. Per esempio:

```
camera {
location <3,5,-10>
sky <1,1,0>
look_at <0,2,1>
}
```

Questa frase permette a POV-Ray di ruotare lateralmente la macchina fotografica fino a che il suo lato superiore non è perpendicolare al vettore sky. Immagina che questo vettore sia un'antenna che punta fuori dalla cima della macchina fotografica. Quindi POV-Ray usa il vettore sky come asse di rotazione e come asse rispetto al quale inclinarsi fino a raggiungere l'allineamento con il vettore sky. In effetti tu stai dicendo a POV-Ray di assumere che il cielo non è direttamente sopra la camera fotografica. È da notare che il vettore sky (cielo) deve apparire prima del vettore look_at.
Il vettore sky non ha effetto da solo. Modifica solamente il modo con cui il vettore look_at orienta la macchina fotografica, il valore di default per sky è <0, 1, 0> (che orienta la macchina fotografica in maniera parallela al suolo).

7.4.4.3 Il Vettore Direzione

Il vettore direzione (parola chiave direction) dice a POV-Ray la direzione iniziale lungo la quale puntare la macchina fotografica prima di muoverla con la parola chiave look_at o con il vettore rotate (il valore di default è direction <0, 0, 1>). Può anche essere usato per controllare l'ampiezza (orizzontale) della visuale in alcuni tipi di proiezione. Dovrebbe tuttavia essere più semplice usare la parola chiave angle.
Se stai usando un 'obiettivo' grandangolo, panoramico o a proiezione cilindrica dovresti usare un vettore direzione unitario (di lunghezza 1) per evitare strani risultati.
La lunghezza del vettore non ha invece importanza se usi uno dei seguenti tipi di proiezione: ortografico, fisheye o omnimax.

7.4.4.4 Angolo (Angle)

La parola chiave angle specifica l'angolo di ampiezza della visuale (orizzontale) della macchina fotografica espresso in gradi. Anche se è possibile usare il vettore direzione per determinare tale angolo, è molto più facile usare la parola chiave angle.
Il calcolo necessario per passare da un metodo all'altro è descritto di seguito. Gli stessi calcoli sono usati per determinare la lunghezza del vettore direzione ogni volta che si incontra la parola chiave

`angle`.
L'angolo della visuale è convertito nella lunghezza del vettore `direction` e viceversa, usando la formula:

```
angle = 2 * arctan(0.5 * lungh_right / lungh_direction)
```

dove `right_length` e `direction_length` sono rispettivamente la lunghezza dei vettori `right` e `direction`, e *arctan* è l'inverso della funzione tangente (arco - tangente).
Per mezzo di questa formula è possibile calcolare la lunghezza del vettore `direction` per un dato angolo di visuale e un determinato vettore `right`.

Semplicemente invertendo la formula si può calcolare la lunghezza del vettore direzione, dati `angle` e `right`.

```
Lungh_direction = 0.5 * lungh_right / tan(angle / 2)
```

7.4.4.5 I Vettori Up e Right

Le direzioni dei vettori `up` e `right`(insieme con il vettore `direction`) determinano l'orientazione della macchina fotografica nella scena. Sono determinate implicitamente dai valori di default di:

```
right 4/3*x
up y
```

oppure dal parametro `look_at` (in combinazione con `location`). Le direzioni specificate esplicitamente da `up` e `right` sono sovrascritte da ogni seguente impostazione del vettore `look_at`.
Mentre alcuni tipi di macchina fotografica ignorano la lunghezza di questi vettori altri li utilizzano per ottenere informazioni circa le impostazioni della macchina fotografica.
La lista seguente spiega il significato dei vettori `up` e `right` per ogni tipo di macchina fotografica. Poiché la direzione dei vettori è sempre usata per descrivere l'orientazione della macchina fotografica, l'utilizzo del vettore `direction` non sarà spiegato di nuovo.

Proiezione prospettica: La lunghezza dei vettori `up` e `right` è usata per impostare le dimensioni ed il rapporto altezza / larghezza della visuale, come viene descritto in dettaglio nella sezione "Rapporto Altezza / Larghezza". Poiché il campo visivo dipende dalla lunghezza del vettore direzione (regolato implicitamente dalla parola chiave `angle`, o esplicitamente da `direction`) e dalle lunghezze dei vettori `up e right`, si devono scegliere con attenzione i valori con cui si possono ottenere i risultati desiderati.

Proiezione ortografica: la lunghezza dei vettori `up e right` fissano la dimensione del campo visivo indipendentemente dalla lunghezza del vettore direzione, che non è utilizzato dalla macchina fotografica *ortografica*. Di nuovo il rapporto tra le lunghezze dei vettori è usato per determinare il rapporto altezza / larghezza.

Proiezione fisheye: i vettori `up` e `right` sono usati per determinare il rapporto altezza / larghezza.

Proiezione ad angolo ultra ampio (*ultra wide angle*): i vettori `up` e `right` funzionano in modo analogo a quanto visto per la proiezione prospettica.

Proiezione Omnimax: questa proiezione è un fisheye da 180 gradi il cui angolo visivo è ridotto nella direzione verticale. Nella realtà questo tipo di proiezione è usato per fare film che possono essere proiettati nei cinema con sistema Omnimax. L'immagine risulterà più o meno ellittica. La parola chiave `angle` non è usata in questo tipo di proiezione.

Proiezione panoramica: i vettori `up` e `right` funzionano in modo analogo a quanto visto per la proiezione prospettica.

Proiezione cilindrica: nelle proiezioni di tipo 1 e 3 l'asse del cilindro giace lungo il vettore `up` e la sua larghezza è determinata dalla lunghezza del vettore `right` , ma può essere modificata anche dal vettore `angle`. Nella proiezione di tipo 3 il vettore `up` determina quante unità è alta l'immagine. Per esempio, se hai definito `up 4*y` sulla tua macchina fotografica posizionata all'origine, solo i punti da y=2 a y= -2 sono visibili. Tutti i raggi visivi sono perpendicolari all'asse y. Per le proiezioni di tipo 2 e 4, il cilindro è disposto lungo il vettore `right`. I raggi visivi per il tipo 4 sono perpendicolari al vettore `right`.

Si deve notare che i vettori `up`, `right` e `direction` dovrebbero sempre rimanere perpendicolari l'un l'altro o l'immagine risulterebbe distorta. Se ciò non avviene, POV-Ray visualizza un messaggio di avvertimento.

7.4.4.5.1 Rapporto Altezza / Larghezza

I due vettori `up` e `right` definiscono insieme il rapporto tra altezza e larghezza dell'immagine. I valori di default, che sono <0,1,0> per `up` e <1.33,0,0> per right risultano in un rapporto 4:3 tra altezza e larghezza, che è il rapporto tra le dimensioni tipico del monitor di un computer. Se vuoi ottenere un'immagine stretta e alta o una bassa, larga e panoramica o perfettamente quadrata, non hai che da agire su questi vettori per ottenere la proporzione appropriata.

La maggior parte delle modalità video e delle stampanti grafiche usano pixel perfettamente quadrati. Per esempio le modalità video dei Macintosh ed anche quelli delle modalità SVGA (640x840, 800x600 e 1024x768) dei personal computer IBM - compatibili adottano pixel quadrati. Quando il metodo di visualizzazione (qualunque sia) che adotti fa uso di pixel quadrati, l'altezza e la larghezza dell'immagine, che imposti con i parametri `+W` e `+H` dovrebbero essere nello stesso rapporto in cui sono `right` e `up`.

Non tutti i sistemi video usano pixel quadrati. Per esempio la modalità IBM/VGA a 320*200 pixel e il display dei computer Amiga a 320x400 pixel non utilizzano pixel quadrati, ma forniscono ancora un'immagine il cui rapporto altezza/larghezza è di 4 a 3. Quindi le immagini da visualizzare con questi hardware dovrebbero ancora utilizzare il rapporto 4/3 tra i vettori `up` e `right` ma le impostazioni di `+W` e `+H` non mantengono questo rapporto.
Per esempio:

```
camera {
location <3,5,-10>
up <0,1,0>
right <1,0,0>
look_at <0,2,1>
}
```

Questo specifica un'immagine perfettamente quadrata. Su un sistema a pixel quadrati come SVGA dovreste usare i parametri `+W` e `+H` come `+W480 +H480` o `+W600 +H600`. Comunque con la

modalità video dell'Amiga (a pixel *non quadrati*) 320x400 dovreste usare `+W240 +H400` per ottenere un'immagine quadrata.

7.4.4.5.2 Chiralità

N.d.T. Chiralità è un termine chimico che deriva dal greco 'chiròs', che significa 'mano', ed è usato per descrivere la diversità di due oggetti simili, ma non uguali, in quanto uno è l'immagine speculare dell'altro, come per esempio la mano destra e la mano sinistra.

Il vettore `right` indica anche la direzione di destra della macchina fotografica, indica cioè a POV-Ray *da che parte è* il lato destro del tuo schermo. Il **segno** del vettore destro può essere usato per determinare il verso del sistema di coordinate utilizzato.
Il vettore `right` di default è :

```
right <1.33, 0, 0>
```

Che significa che la direzione delle x positive è a destra. Questo sistema di assi viene chiamato sistema di coordinate *sinistrorso* perché puoi usare la mano sinistra per ricordare dove si trovano gli assi. Disponi la mano sinistra con il palmo rivolto verso destra. Punta il pollice in alto, l'indice davanti a te e piega le altre dita verso destra; queste puntano verso la direzione +x, il pollice punta nella direzione +y e l'indice nella direzione +z.

Per utilizzare un sistema di coordinate destrorso, diffuso in alcuni programmi di CAD e altri programmi di raytracing, puoi ottenere lo stesso riferimento utilizzando la mano destra. Il pollice è ancora puntato in alto. Nella direzione +y, l'indice punta ancora verso la direzione +z ma le altre dita adesso indicano che la posizione +x è a sinistra. Ciò vuol dire che il lato destro dello schermo è adesso nella direzione -x. Per dire a POV-Ray di comportarsi in questo modo potete usare un valore negativo nella componente x del vettore `right` come in questo esempio:

```
right <-1.33, 0, 0>
```

Poiché avere le coordinate x che crescono verso sinistra non ha molto senso su uno schermo a due dimensioni, adesso ruota tutto il sistema di 180° usando un valore positivo nella componente z del vettore `location` della tua macchina fotografica. Dovresti ottenere qualcosa di questo genere:

```
camera {
location <0,0,10>
up <0,1,0>
right <-1.33,0,0>
look_at <0,0,0>
}
```

Adesso quando esegui i tuoi esercizi di 'aerobica per raytracing', come è spiegato nella sezione "Capire il Sistema di Coordinate di POV-Ray", usi la mano destra per determinare la direzione delle rotazioni.

In un sistema di assi a due dimensioni, la direzione positiva dell'asse x è sempre rivolta verso destra, e la direzione positiva dell'asse delle y è sempre rivolta in alto. Le due diverse possibilità (di usare cioè sistemi di riferimento destrorsi o sinistrorsi) sorgono dal problema di definire se l'asse delle z punta all'interno o all'esterno dello schermo, e quale degli assi nel modello al computer si riferisce all'altezza nel modo reale.

I sistemi di CAD per architettura, come AutoCAD, tendono ad usare un'orientazione 'dall'alto' (in

originale, *God's Eye*, che significa letteralmente 'Occhio di Dio') poiché l'asse z corrisponde all'elevazione e quindi alla direzione verticale per il modello. Questo approccio ha un senso se tu sei un architetto che osserva il progetto di un edificio sullo schermo di un computer. Z significa 'in alto', l'asse z è rivolto *verso* l'osservatore, mentre gli assi x ed y sono rispettivamente orizzontale e verticale sullo schermo.

Sistemi di rendering, come POV-Ray, tendono a considerarti come *parte* della scena : l'osservatore è rivolto verso lo schermo come un fotografo che si trova nella scena. In questo caso, la direzione verticale è quella delle y (come nel mondo reale), e l'asse delle x è rivolto verso destra, quidni la direzione delle z deve essere opposta all'osservatore (punta all'interno del monitor). Questo è il sistema di coordinate sinistrorso.

7.4.4.6 Trasformazioni sulla Macchina Fotografica

Le trasformazioni di traslazione e rotazione possono riposizionare la macchina fotografica dopo che questa è stata definita. Per esempio:

```
camera {
location < 0, 0, 0>
direction < 0, 0, 1>
up < 0, 1, 0>
right < 1, 0, 0>
rotate <30, 60, 30>
translate < 5, 3, 4>
}
```

In questo esempio, la macchina fotografica dopo essere stata creata, è ruotata di 30 gradi sull'asse x, di 60 sull'asse y e di 30 sull'asse delle z, e poi traslata in un altro punto nello spazio.

7.4.5 Identificatori della Macchina Fotografica

Puoi dichiarare, se vuoi, diversi identificatori della macchina fotografica. Questo rende più facile cambiarle rapidamente. Per esempio:

```
#declare Teleobiettivo =
camera {
location -z*100
angle 3
}

#declare Grandangolo =
camera {
location -z*50
angle 15
}

camera {
Teleobiettivo // edita questa linea per cambiare l'obiettivo
look_at Punto_di_mira
}
```

7.5 Oggetti

Gli oggetti sono ciò che costituisce la tua scena. C'è un gran numero di oggetti diversi supportati da POV-Ray : primitive solide finite, superfici finite, primitive polinomiali solide infinite e sorgenti luminose. E' supportata anche la Geometria Solida Costruttiva, o Booleana (CSG, *Constructive Solid Geometry*).

La sintassi di base di un oggetto è una parola chiave che ne descrive il tipo, alcuni valori decimali, vettori o altri parametri che ne definiscono ulteriormente la posizione e/o la forma ed alcuni modificatori facoltativi che influiscono sull'oggetto, come texture, pigmento, normali, finitura, bounding, clipping e trasformazioni.

La **texture** descrive il materiale da cui l'oggetto sembra ricavato. Le texture sono il risultato delle combinazioni di pigmenti, normali, finiture ed halo.

Pigmento è il colore o i motivi di più colori contenuti nel materiale.

Le **Normali** sono un metodo per *simulare* varie irregolarità della superficie di un oggetto, come rigonfiamenti, ammaccature, increspature oppure ondulazioni tramite perturbazioni del vettore normale alla superficie.

La **Finitura** descrive le proprietà riflessive e rifrattive di un materiale.

Halo è usato per descrivere l'interno dell'oggetto, e più precisamente riempiono l'oggetto con microscopiche particelle per ottenere effetti speciali come fuoco, fumo, nebbia o altro ancora.

Gli oggetti delimitanti (*bounding shapes*) sono solidi finiti ed **invisibili** che racchiudono oggetti complessi entro forme più semplici, per accelerarne il rendering.

Altri oggetti, detti *clipping shapes* 'ritagliano' parti di un oggetto per renderne visibile l'interno.

Le **trasformazioni** dicono a POV-Ray come muovere, ruotare o ridimensionare l'oggetto e/o le texture nella scena.

7.5.1 Oggetti Vuoti e Oggetti Solidi

E' molto importante capire pienamente il concetto di *oggetto vuoto* che POV-Ray usa per comprendere meglio come funzionano le halo e la trasparenza.
Gli oggetti possono essere solidi, vuoti o riempiti con piccole particelle. Un oggetto solido è fatto dal materiale specificato nelle frasi di pigmento e di finitura (e in qualche modo anche dalle frasi che specificano le normali). Di default ogni oggetto viene considerato solido. Se viene assegnata una texture 'pietra' ad una sfera, si otterrà una palla completamente fatta di pietra, come se fosse stata tagliata da un blocco di pietra più grande. Una sfera di vetro sarà anch'essa completamente di vetro.
Bisogna notare che gli oggetti solidi sono tali solo concettualmente. Per esempio se si potesse tagliare via una parte della sfera si noterebbe che l'interno è vuoto e che l'oggetto ha solo una superficie molto sottile.
Questo è il contrario del concetto di *solido*, ma viene adottato da POV-Ray : si presume che in un solido anche l'interno della sfera sia pieno dello stesso materiale di cui la sfera è composta.
In questo modo non c'è ulteriore posto per altre particelle come quelle usate per le halo e per la nebbia.
Gli oggetti cavi sono creati aggiungendo la parola chiave `hollow` nella frase che descrive l'oggetto (vedi "Hollow (Vuoto)"). Un oggetto cavo è costituito da una superficie molto sottile fatta da materiale, pigmento, finitura e normali. L'interno dell'oggetto è vuoto, normalmente conterrebbe

molecole di aria.
Un oggetto cavo può essere riempito da particelle aggiungendo l'atmosfera o la nebbia alla scena o aggiungendogli una halo. E molto importante capire che per riempire un oggetto con qualunque tipo di particelle è necessario specificare la parola chiave `hollow`.

7.5.1.1 Tranelli degli Aloni

E' necessario avvertirvi che in questa versione di POV-Ray c'è un tranello per quanto riguarda gli oggetti pieni e vuoti.
Per essere in grado di mettere un alone dentro ad oggetti solidi (questo vale anche per l'atmosfera e la nebbia) è necessario fare un test per ogni raggio che passa attraverso la halo. Se questo raggio passa attraverso un oggetto solido non sarà calcolata l'halo. Questo è esattamente quello che ci saremmo aspettati. Il problema nasce quando il raggio della macchina fotografica si trova *all'interno* di un oggetto solido. In questo caso il raggio sta già passando attraverso un oggetto non cavo e quindi, se l'oggetto che contiene l'alone è colpito da questo raggio e questo oggetto è cavo, l'alone non sarà calcolato. Non c'è nessun modo di differenziare questi due casi.
POV-Ray deve determinare se la camera è all'interno di un oggetto non cavo *prima* di tracciare i raggi della macchina fotografica, per essere in grado di renderizzare correttamente gli aloni quando la macchina fotografica è all'interno dell'oggetto contenitore. Non c'è modo per aggirare questo problema.
La soluzione al problema (che si ripete spesso con oggetti infiniti come i piani) è di rendere cavi anche questi oggetti. Così il raggio passerà attraverso oggetti cavi, colpirà l'oggetto contenitore e sarà possibile calcolare l'halo. Quanto detto sopra vale anche per la nebbia e per l'atmosfera.

7.5.1.2 Tranelli della rifrazione

Questo tranello coinvolge gli oggetti che possono dar luogo a rifrazione (e quelli trasparenti ma non rifrangenti).
Immaginate di voler creare un oggetto che è fatto in parte di vetro e in parte di pietra, usando una frase `merge{...}`per non vedere le superfici interne.

```
merge {
sphere { <-1,0,0>, 2 texture { Pietra } }
sphere { <+1,0,0>, 2 texture { Vetro } }
}
```

Cosa c'è di sbagliato qui? Il problema è che non c'è modo di distinguere di quale materiale sarà l'interno dell'oggetto. Questo non è un problema solo di POV-Ray , ma generale. Non è possibile definire l'interno di un oggetto in un modello basato sulle superfici. Sarebbe necessario creare regole molto complesse per poter stabilire di quale materiale è costituito l'oggetto all'interno. Sarà fatto di pietra ? Di vetro ? Di una strana mistura fra i due ? E se fosse fatto per metà di pietra e per metà di vetro ? Come risulterà il passaggio tra i due materiali ?
Non sarai in grado di rispondere a nessuna di queste domande guardando semplicemente l'oggetto. Ti saranno necessarie maggiori informazioni.
Se vuoi creare un oggetto metà di pietra e metà di vetro usa la seguente sintassi :

```
union {
intersection {
sphere { <-1,0,0>, 2 }
plane { x, 0 }
texture { Pietra }
}
intersection {
```

```
sphere { <+1,0,0>, 2 }
plane { x, 0 inverse }
texture { Vetro }
}
}
```

questo esempio è corretto e non incorre in ulteriori tranelli perché c'è l'unione di due oggetti : uno fatto di pietra e uno di vetro.
Non dovresti mai usare oggetti il cui interno non sia ben definito, non ci devono essere texture diverse e con diversi indici di rifrazione (o diversi valori di trasparenza) nello stesso oggetto. Queste eventualità possono funzionare solo per lo strato più basso di una texture stratificata.
Vedi anche "Tranelli degli Aloni".

7.5.2 Primitive Solide Finite

Ci sono dodici tipi di primitive finite : blob, parallelepipedi, coni, cilindri, frattali, height field, lathe, sfere, superellissoidi, superfici di rotazione, tori e oggetti composti col testo.
Questi oggetti hanno un interno ben definito e possono essere usati nelle operazioni CSG (vedi il paragrafo "Geometria Solida Costruttiva"). sono finiti ed è possibile applicare loro il bounding automatico. Come tutti gli oggetti possono essere spostati, ruotati e scalati.

7.5.2.1 Blob

I blob sono oggetti versatili ed interessanti. Matematicamente sono superfici equipotenziali di un campo scalare, cioè, la loro superficie è definita dalla forza del campo in ciascun punto. Se la forza in un punto è uguale al valore di soglia, quel punto si trova sulla superficie, negli altri casi no.
Immagina ogni componente di un blob come un oggetto che galleggia nello spazio. Quest'oggetto è riempito da un campo che ha il suo massimo al centro dell'oggetto e che decade a zero sulla superficie. La forza del campo di tutte le componenti viene sommata per formare il campo complessivo del blob. POV-Ray cerca i punti dove il campo ha un valore predefinito (cioè quello del valore di soglia, detto *threshold*). Tutti questi punti formeranno la superficie dell'oggetto. Punti con un valore della forza maggiore del valore di soglia sono all'interno dell'oggetto, mentre i punti con un valore di forza minore di quello di soglia, si trovano all'esterno. C'è un modo più semplice di immaginarsi i blob. Si possono considerare come unioni di componenti flessibili che si attraggono o si respingono l'un l'altro. La superficie dei componenti si estende gradualmente fino a congiungerli, come se fossero fatti di miele, o di una qualche sostanza vischiosa.
Un blob è costituito da componenti sferiche e cilindriche, ed è definito come segue :

```
blob {
threshold VALORE_DI_SOGLIA
cylinder { <ESTREMITA'1>, <ESTREMITA'2>, RAGGIO, [ strength ]
FORZA }
sphere { <CENTRO>, RAGGIO, [ strength ] FORZA }
[ component FORZA, RAGGIO, <CENTRO> ]
[ hierarchy on/off]
[ sturm ]
}
```

La parola chiave `threshold` determina il valore totale della forza del campo ricercato da POV-Ray. Seguendo il raggio di luce nello spazio, ed osservando come questo è influenzato da ogni componente, POV-Ray determina i punti dello spazio in cui la forza del campo è uguale al valore di soglia. Il seguente elenco mostra alcuni aspetti importanti del valore di soglia :

1) Il valore di soglia `threshold` deve essere positivo.
2) Un componente scompare se il valore di soglia è maggiore della sua forza.
3) Più grande è il valore di soglia, più la superficie sarà vicina al centro dei componenti.
4) Minore è il valore di soglia, più la superficie sarà vicina alla superficie dei componenti.

I componenti cilindrici sono specificati dalla parola chiave `cylinder` che definisce un cilindro formato dalle estremità nei punti `<ESTREMITA'1>` ed `<ESTREMITA'2>` e dal raggio. Le estremità del cilindro sono emisferiche. I componenti sferici sono invece specificati dalla parola chiave `sphere` che definisce il centro ed il raggio. Ciascun componente può essere spostato, ruotato, scalato, e gli può essere assegnata una texture. La sintassi completa per i componenti sferici e cilindrici è :

```
sphere { <CENTRO>, RAGGIO, [strength] FORZA
[ translate <VETTORE> ]
[ rotate < VETTORE > ]
[ scale < VETTORE > ]
MODIFICATORI_DI_TEXTURE
}

cylinder { <END1>, <END2>, RAGGIO, [strength] FORZA
[ translate < VETTORE > ]
[ rotate < VETTORE > ]
[ scale < VETTORE > ]
MODIFICATORI_DI_TEXTURE
}
```

Scalando lungo uno o due assi un componente sferico, si possono creare componenti ellissoidali. La parola chiave `component` dà un componente sferico ed è usata solo per mantenere la compatibilità con le precedenti versioni di POV-Ray. Il parametro `strenght` è un valore decimale che specifica la forza del campo nel centro dell'oggetto. La forza può essere positiva o negativa. Un valore positivo farà sì che un componente *attragga* gli altri componenti, una forza negativa glieli farà respingere.
I componenti di blob diversi non si influenzano a vicenda.
Si deve tenere a mente che :

1) Il valore della forza può essere positivo o negativo. Zero non è un valore accettabile perché in questo caso non ci sarebbe alcun campo di forza, e quindi è come se il componente non fosse stato aggiunto.
2) Se il valore della forza è positivo, POV-Ray aggiunge la forza del componente allo spazio attorno al suo centro. Se la forza aggiunta è maggiore del valore di soglia, la superficie del componente è visibile.
3) Se il valore della forza è negativo, POV-Ray sottrae il campo del componente dallo spazio circostante. Questo ha effetto solo se ci sono componenti positive vicine. Quello che succede è che la superficie delle componenti positive che lo circondano viene respinta a partire dal centro del componente negativo.

I componenti di ogni blob sono delimitati automaticamente da una gerarchia di sfere, per accelerare i test di intersezione raggio/oggetto e le altre operazioni. Usando la parola chiave facoltativa `hierarchy` si può disattivare questa funzione.
Un esempio di un blob a tre componenti è :

```
blob {
threshold 0.6
sphere { <.75, 0, 0>, 1, 1 }
sphere { <-.375, .64952, 0>, 1, 1 }
sphere { <-.375, -.64952, 0>, 1, 1 }
scale 2
}
```

Se il tuo blob è costituito da un unico componente, allora la superficie visibile è come quella del componente, cioè una sfera o un cilindro la cui superficie si trova all'interno i quella specificata. L'esatta posizione della superficie può essere determinata usando l'equazione del blob che si trova qui sotto (probabilmente non avrai mai bisogno di saperla, i blob sono più utili per il loro aspetto che per modellazione precisa).
Per quelli più portati per la matematica, questa è la formula che POV-Ray usa per creare i blob. Non è necessario capirla per usarli. La formula usata per un blob composto da un unico componente è :

```
densità = forza*(1-raggio^2)^2
```

7.5.2.2 Parallelepipedo

Un semplice parallelepipedo può essere definito elencandone due vertici in questo modo :

```
box { <VERTICE1>, <VERTICE2> }
```

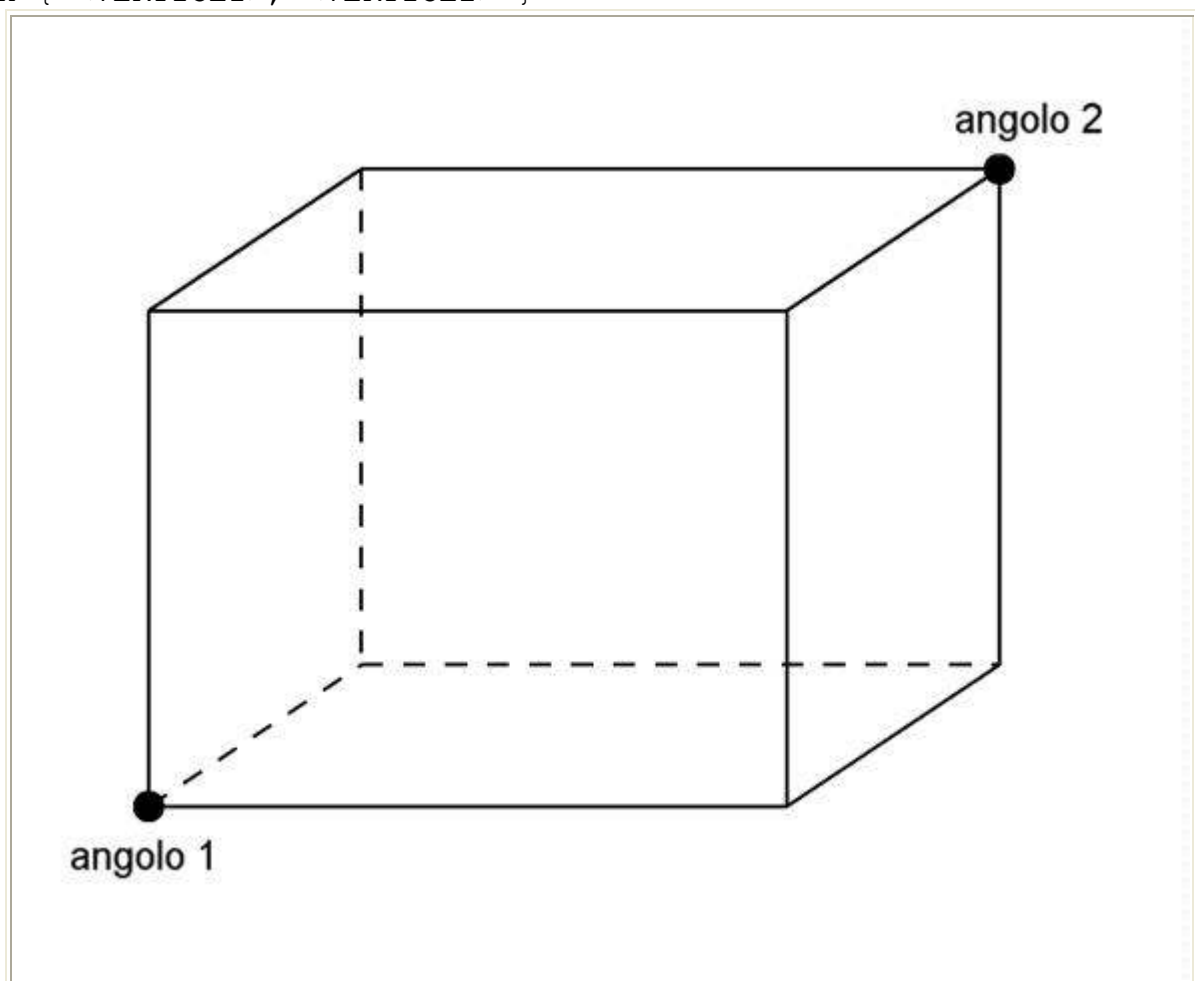



Fig. 198-Parallelepipedo

Dove <VERTICE1> e <VERTICE2> sono vettori che definiscono le coordinate x, y, z di due vertici opposti del parallelepipedo. E' da notare che tutti i parallelepipedi hanno le facce parallele agli assi coordinati. Possono essere in seguito ruotati usando la parola chiave `rotate`.
Ogni elemento del vettore <VERTICE1> deve essere sempre minore del corrispondente elemento del vettore <VERTICE2>. Se un valore di <VERTICE1> è maggiore del corrispondente valore di <VERTICE2>, il parallelepipedo non apparirà nella scena.
I parallelepipedi sono calcolati velocemente da POV-Ray e possono quindi essere usati efficacemente come solidi delimitanti (*bounding shapes*).

7.5.2.3 Cono

Un cono di altezza finita, o un tronco di cono (un cono con la punta tagliata) può essere definito così :

```
cone {
<CENTRO_BASE>, RAGGIO_BASE, <CENTRO_PUNTA>, RAGGIO_PUNTA
[ open ]
}
```

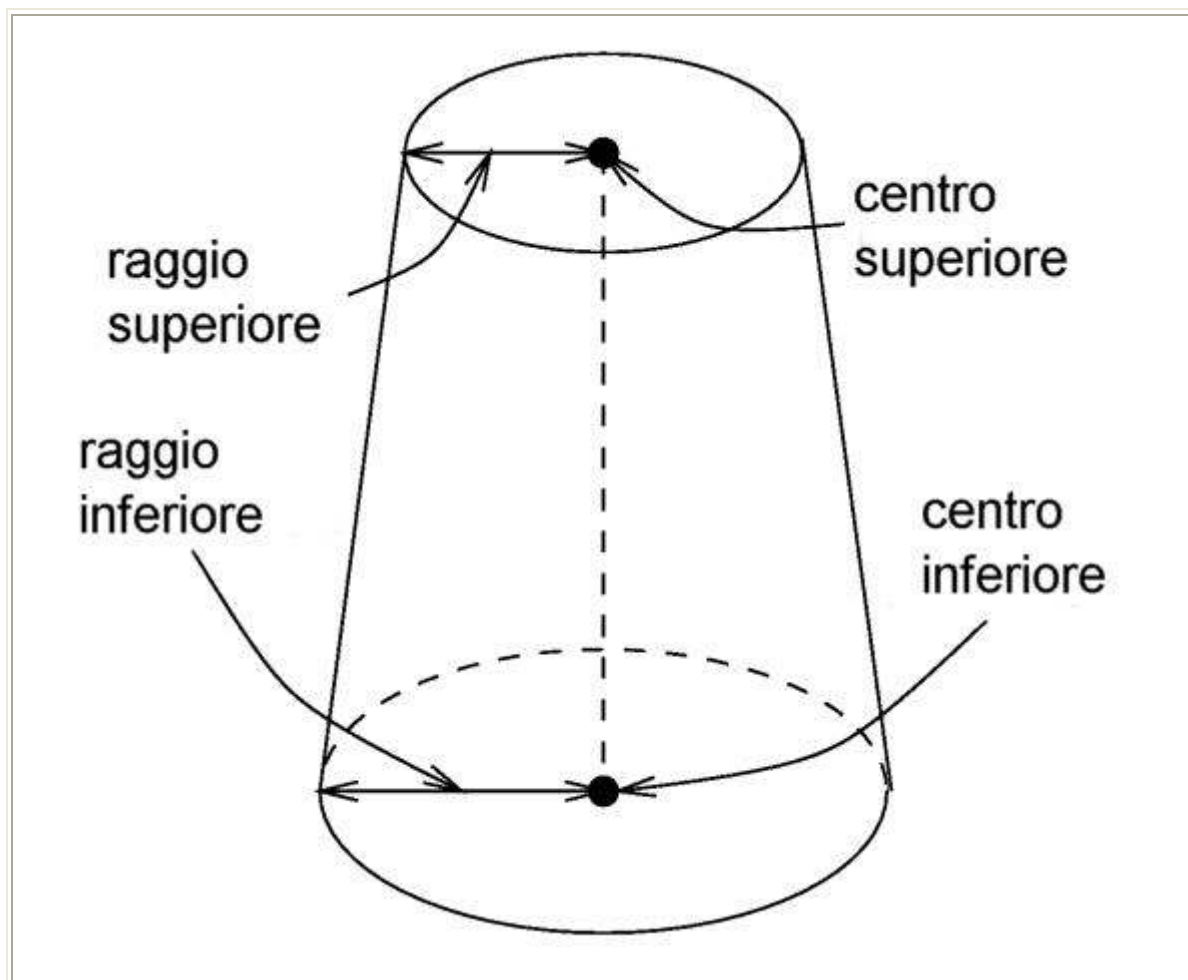



Fig. 199-Tronco di cono

Dove <CENTRO_BASE> e <CENTRO_PUNTA> sono vettori che definiscono le coordinate x, y e z del centro della base e del centro della punta del cono e `RAGGIO_BASE` e `RAGGIO_PUNTA` sono valori decimali per i corrispondenti raggi.
Normalmente le estremità del cono sono chiuse da piani paralleli fra loro e perpendicolari all'asse

del cono. Aggiungendo la parola chiave `open` (facoltativa) dopo `<RAGGIO_PUNTA>` si rimuoveranno questi piani dando un 'tubo' conico dalla forma di un megafono.

7.5.2.4 Cilindro

Un cilindro di altezza finita chiuso alle estremità da facce parallele può essere definito così :

```
cylinder {
<CENTRO_BASE_INF>, <CENRO_BASE_SUP>, RAGGIO
[ open ]
}
```

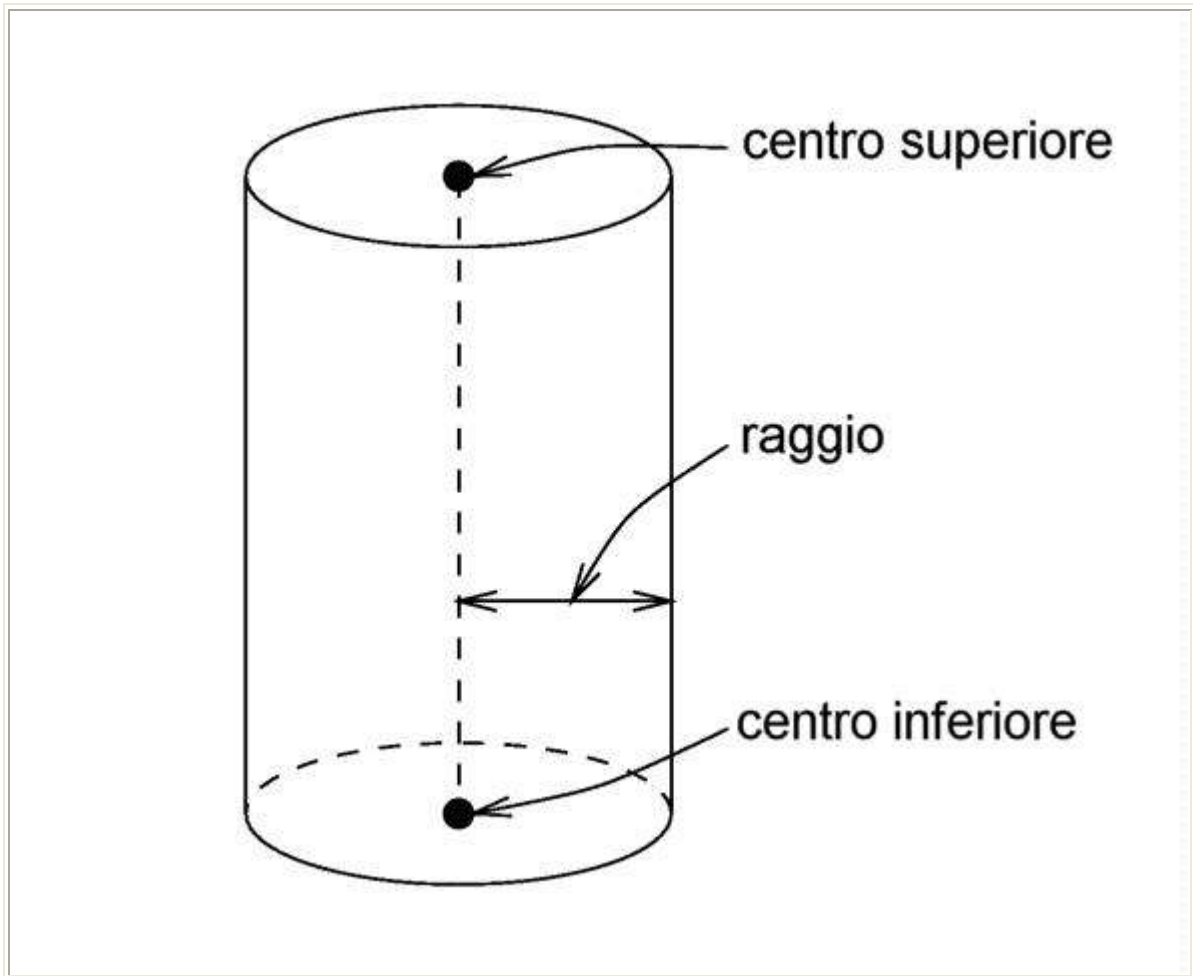



Fig. 200-Cilindro

Dove `<CENTRO_BASE_SUP>` e `<CENTRO_BASE_INF>` sono vettori che definiscono le coordinate x,y delle basi superiore e inferiore del cilindro, e `RAGGIO` è un numero decimale che specifica il raggio del cilindro. Normalmente le estremità di un cilindro sono chiuse da piani paralleli fra di loro e perpendicolari all'asse del cilindro. Aggiungendo la parola chiave `open` (facoltativa) dopo `RAGGIO` si toglieranno questi due piani e ne risulterà un tubo cavo.

7.5.2.5 Campi di Quota (Height Field)

Gli height field sono oggetti veloci ed efficienti che sono generalmente usati per ottenere montagne o altre superfici che si innalzano partendo da un oggetto *mesh* composto da centinaia di triangoli. La sintassi per gli height field è :

```
height_field {
TIPO_FILE "NOMEFILE"
[ hierarchy on/off ]
[ smooth on/off ]
[ water_level DECIMALE]
}
```

L' height field è essenzialmente un riquadro ampio e profondo un'unità, con il lato superiore dall'aspetto 'montuoso'. L'altezza delle montagne in ciascun punto è calcolata dal numero del colore o dal suo indice nella tavolozza per il punto corrispondente in un'immagine. L'altezza massima è 1, che corrisponde al colore di indice massimo nell'immagine (ad esempio, per un'immagine GIF a 256 colori, il valore massimo è dato dal colore n° 255).

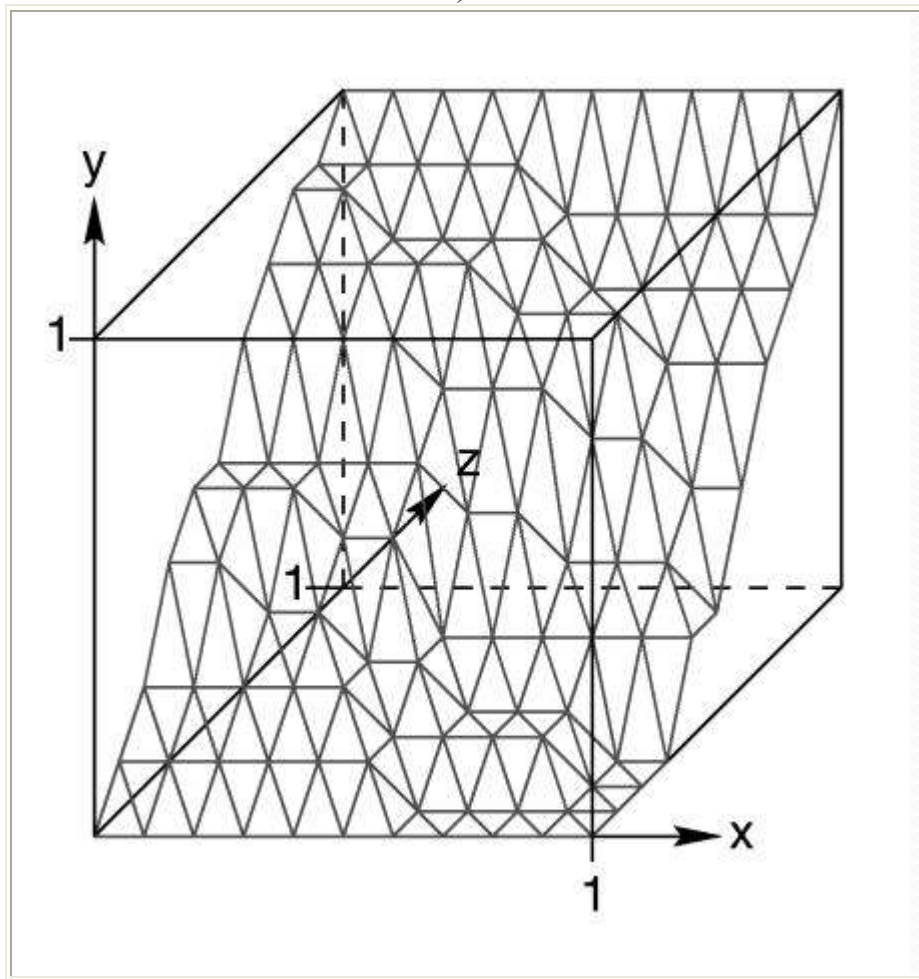



Fig. 201-Campi di quota

L'insieme di triangoli corrisponde direttamente ai punti nel file immagine. Ogni riquadro formato da quattro pixel adiacenti è suddiviso in due triangoli. Un immagine con risoluzione N*M punti, contiene (N-1)*(M-1) riquadri che sono divisi in 2*(N-1)*(M-1) triangoli.

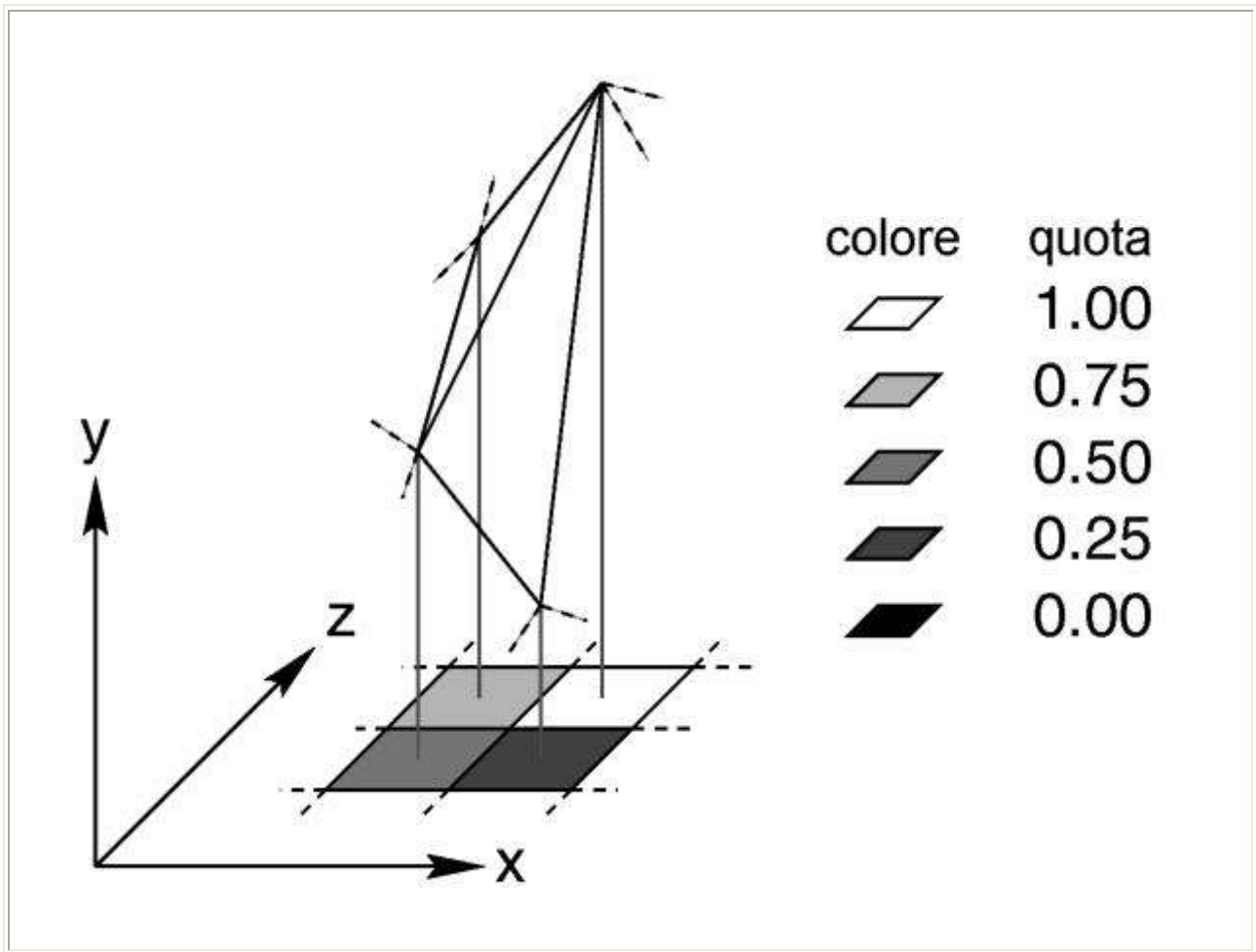



Fig. 202-Quote in funzione dei colori

La risoluzione dell'height field è influenzata da due fattori : la risoluzione dell'immagine e il numero di colori. La dimensione dell'immagine determina la risoluzione dell'height field nelle direzioni x e z. Un'immagine più grande fa uso di un maggior numero di triangoli ed ha quindi un aspetto più uniforme. Il numero di colori determina la risoluzione lungo l'asse delle y. Un height field creato a partire da un immagine ad 8 bit/pixel può avere 256 livelli di quota diversi, mentre uno creato a partire da un'immagine a 16 bit/pixel può avere 65536 livelli di quota diversi. In questo modo, il secondo height field avrà un aspetto molto più uniforme del primo, se viene creato e manipolato in maniera appropriata.

La dimensione dell'immagine non influisce sulle dimensioni dell'height field, che rimangono comunque 1*1 (a meno di non scalarlo). Immagini ad alta risoluzione daranno più triangoli, non un height field più grande.
Ci sono sei o forse sette tipi di file immagine che possono definire un height field :

```
height_field { gif "filename.gif" }
height_field { pgm "filename.pgm" }
height_field { png "filename.png" }
height_field { pot "filename.pot" }
height_field { ppm "filename.ppm" }
height_field { sys "filename.???" }
height_field { tga "filename.tga" }
```

Il file immagine che si usa per creare un height field può essere in formato GIF, TGA, POT, PNG,

PGM, PPM e se disponibile anche un formato specifico per la piattaforma, come BMP per Windows oppure Pict per Macintosh. I formati GIF, PNG, PGM e quello specifico per sistema (ovviamente in dipendenza dal sistema) sono gli unici che possono essere creati usando un programma di disegno. Anche se esistono programmi di disegno che creano file .TGA, non saranno molto utili per creare lo speciale formato TGA a 16 bit/pixel che POV-Ray usa (vedi più avanti, ed il paragrafo "HF_Gray_16").
In un formato di immagine come GIF che usa la tavolozza dei colori, il *numero di colore* è il numero progressivo che il colore di un certo pixel ha nella tavolozza. Usa un programma di disegno per vedere la tavolozza di colori dell'immagine : Il primo colore è contrassegnato dal numero zero, il secondo ha il numero 1 e così via. L'ultimo colore della tavolozza è il numero 255.
Le parti dell'immagine che usano colori il cui numero d'ordine è basso risulteranno in parti più basse dell'height field. Parti dell'immagine che usano colori il cui numero d'ordine è più alto, daranno le parti più alte.
Gli height field creati da immagini GIF possono avere solo 256 livelli di quota diversi perché il numero massimo di colori in un'immagine GIF è 256. Il colore corrispondente ad un determinato indice non influenza l'altezza del punto. Il colore 0 potrebbe essere rosso, blu, nero o arancione ma l'altezza di tutti i punti che usano il colore 0 sarà sempre 0. Il colore 255 potrebbe essere azzurro, giallo, bianco o verde ma tutti i punti che corrispondono al colore 255 saranno sempre ad altezza 1.
Si possono creare immagini GIF adatte per generare height field con programmi di disegno o con programmi di generazione frattale delle immagini come **Fractint**. Si può trovare **Fractint** in quasi tutti i posti in cui si trova POV-Ray.
Un file POT è fondamentalmente un file GIF con una tavolozza di colori di 16 bit/pixel. Il numero massimo di colori in un file POT è 65536. Quindi un height field generato Da un file POT può avere fino a 65536 livelli diversi. Ciò rende possibile avere height field molto più dettagliati. In ogni caso l'altezza massima è sempre uno anche se sono possibili più valori intermedi.
Al momento in cui questa guida viene scritta l'unico programma che crea file POT è un programma freeware per PC chiamato **Fractint**. Le immagini POT create con questo programma danno paesaggi fantastici.
I formati TGA e PPM possono essere usati più come strumento di memorizzazione di numeri a 16 bit, che come immagini. Questi formati usano i valori del rosso e del verde di ogni punto per immagazzinare i valori che determinano le quote di un height field. Questi file danno risultati analoghi a quelli dei file POT, ma devono essere generati da appositi programmi. Alcuni programmi possono creare immagini TGA nel formato usato da POV-Ray. Tra questi **Gforge** e **Terrain Maker** (nonché lo stesso POV-Ray, N.d.T.).
Gli height field in formato PNG sono generalmente memorizzati come immagini a toni di grigio in cui il nero corrisponde alle parti più basse e il bianco a quelle più alte. Poiché i file PNG possono memorizzare immagini a toni di grigio fino a 16 bit/pixel gli height field generati da esse saranno dettagliati come quelli ottenuti dalle analoghe immagini in formato TGA e PPM. Poiché sono immagini a toni di grigio, si potranno vedere con un normale visore. **Gforge** può creare height field a 16 bit in formato PNG. Le immagini PNG a colori verranno invece usate come le TGA e le PPM.
Il formato SYS è specifico della piattaforma usata. Vedi la documentazione specifica per dettagli.
Un parametro facoltativo, indicato dalla parola chiave `water_level` può essere aggiunto dopo il nome del file. La parola chiave è seguita da un numero decimale che dice a POV-Ray di tralasciare le parti dell'height field la cui altezza è al di sotto di quel valore. Il valore di default è zero, si possono usare valori tra zero e uno. Ad esempio `water_level .5` fa renderizzare a POV-Ray solo la metà superiore dell'height field. L'altra metà è "sott'acqua" (*water level* significa 'livello dell'acqua') e non si vedrebbe comunque. Questo termine deriva dal normale uso degli height field per renderizzare paesaggi. Si potrebbe infatti usare un height field per creare un'isola e usare un altro oggetto per simulare il mare. Una gran parte dell'height field sarebbe sotto il livello dell'acqua, quindi è stato introdotto il parametro `water_level` che permette a POV-Ray di ignorarne la parte non visibile. La parola chiave `water_level` può servire anche per rimuovere parti indesiderate di

un height field corrispondenti a quote basse. Ad esempio, se hai un'immagine di un frattale su uno sfondo a tinta unita, ed il colore dello sfondo è il colore 0 nella tavolozza, lo si può rimuovere dall'height field specificando `water_level .001`.
Normalmente gli height field hanno un aspetto ruvido ed irregolare, poiché sono costituiti da molti triangoli. Aggiungendo la parola chiave `smooth` POV-Ray può modificare i vettori normali alla superficie dei triangoli in modo tale che le luci e le ombre dei triangoli appaiano più uniformi. Questo può permetterti di utilizzare una minore risoluzione di quella che sarebbe in realtà necessaria quando renderizzi l'height field. Comunque, l'uso dei triangoli smussati aumenterà il tempo di rendering.
Per aumentare le velocità dei test di intersezione raggio/oggetto, è possibile una gerarchia di bounding ad un livello. Per default questa viene sempre usata, ma può essere disattivata per aumentare la velocità di rendering per piccoli height field (generati cioè da immagini a bassa risoluzione).

7.5.2.6 Frattali di Julia

Un frattale di Julia è una 'fetta' tridimensionale di un oggetto quadridimensionale creato generalizzando il procedimento che si usa per creare i classici *insiemi frattali di Julia*. Puoi ottenere una grande quantità di strani oggetti usando questo tipo di frattale, compresi alcuni che avranno l'aspetto di strani ammassi spiraliformi.
La sintassi di un frattale Julia è la seguente :

```
julia_fractal {
PARAMETRO_4D // il valore di default è <1,0,0,0>
[ quaternion | hypercomplex ] // il valore di default è quaternion
[ sqr | cube | exp |
reciprocal | sin | asin |
sinh | asinh | cos | acos |
cosh | acosh | tan | atan |
tanh | atanh | log | pwr(X,Y) ] // il valore di default è sqr
[ max_iteration N_MAX_ITERAZIONI] // il valore di default è 20
[ precision PRECISIONE ] // il valore di default è 20
[ slice NORMALE_4D, DISTANZA ] // i valori di default sono
<0,0,0,1>,0
}
```

Il vettore a quattro dimensioni `PARAMETRO_4D` è il classico parametro `p` della formula iterativa di Julia `f(h)+p`.
Il frattale di Julia è calcolato usando un algoritmo che determina se un punto arbitrario h(0) in uno spazio *quadridimensionale* si trova all'interno o all'esterno dell'oggetto. L'algoritmo richiede che venga generata la sequenza di vettori h(0), h(1)... iterando la formula :

```
h(n+1) = f(h(n)) + p (n = 0, 1, ..., max_iteration-1)
```

dove p è il vettore quadridimensionale del frattale Julia, ed f() è una delle funzioni elencate sopra, specificata dalla presenza della parola chiave corrispondente. Il punto h(0) che inizia la sequenza è considerato all'interno del frattale Julia, se nessuno dei vettori della sequenza esce da una *ipersfera* (sfera a quattro dimensioni) di raggio 4 attorno all'origine, prima che l'iterazione dei numeri raggiunga il valore specificato per `max_iterations`. Aumentando questo valore, alcuni punti escono dall'ipersfera, formando il frattale.
A seconda del vettore di partenza assegnato, l'oggetto 'frattale' non è necessariamente un oggetto aggregato : può accadere che sia una nuvola di polvere frattale. Usando un basso valore di

`max_iterations` è possibile riunire la polvere e creare un oggetto solido. Un valore alto per `max_iterations` è più accurato, ma richiede un maggiore tempo di rendering.
Anche se non sono calcolati accuratamente, gli oggetti ottenuti con un basso valore di `max_iterations` possono comunque essere molto interessanti. Dal momento che l'oggetto matematico descritto da questo algoritmo è quadridimensionale e POV-Ray renderizza oggetti tridimensionali, si deve trovare un modo di ridurre il numero delle dimensioni dell'oggetto da 4 a 3. Questo si ottiene facendo intersecare il frattale quadridimensionale con un piano tridimensionale e proiettando l'intersezione nello spazio tridimensionale. Questo piano è lo spazio tridimensionale perpendicolare a `NORMALE_4D` e distante `DISTANZA` dall'origine. Non sono accettabili valori di 0 per `NORMALE_4D` o per la sua quarta componente.
Puoi entrare in confidenza con la natura quadridimensionale del frattale di Julia usando un'animazione e facendo variare con `clock` il parametro della distanza del piano (`DISTANZA`). Puoi far sì che il frattale appaia dal nulla, cresca e di nuovo svanisca in relazione al variare del parametro distanza, in modo molto simile a ciò che accade alla sezione bidimensionale di un oggetto 3D che passa attraverso un piano. Il parametro `precision` è la tolleranza usata nel calcolo dei punti dell'oggetto. Maggiori valori danno un risultato più accurato, ma un rendering più lento. Usa un valore quanto più possibile basso, evitando però che il frattale si deteriori.
La presenza delle parole chiave `quaternion` o `hypercomplex` determina quale tipo di algebra quadridimensionale è usato per calcolare il frattale. Entrambe sono generalizzazioni quadridimensionali dei numero complessi, ma nessuna soddisfa tutte le proprietà del campo (tutte le proprietà dei numeri reali e complessi, sulle quali molti di noi hanno dormito alle superiori). I quaternioni hanno la moltiplicazione *non commutativa*, mentre i numeri ipercomplessi possono non avere un inverso rispetto alla moltiplicazione per alcuni numeri diversi da zero (è stato dimostrato che non si possono estendere i numeri complessi alle quattro dimensioni mantenendone tutte le proprietà, quindi qualcosa *non può* conservarsi). Entrambi questi metodi sono stati scoperti nel XIX secolo. Tra le due, l'algebra dei quaternioni è molto più nota, ma si potrebbe obiettare che gli ipercomplessi sono più utili per i nostri scopi, dato che le funzioni di valori complessi, come seno, coseno, ecc. possono essere estese ai numeri ipercomplessi in maniera uniforme.

Per i curiosi, le proprietà algebriche di queste due algebre possono essere derivate dalle proprietà della moltiplicazione dei vettori che formano la base canonica dello spazio a quattro dimensioni : 1 = <1,0,0,0>, *i*=< 0,1,0,0>, *j*=<0,0,1,0> e *k*=< 0,0,0,1>. In entrambe le algebre, per ogni *x* vale che 1**x* = *x**1 = *x* (dove 1 è l'elemento neutro rispetto alla moltiplicazione). I vettori base 1 ed *i* si comportano esattamente come i familiari numeri complessi 1 ed *i* in entrambe le algebre.

Le regole di moltiplicazione dei vettori della base, nell'algebra dei quaternioni, sono le seguenti :

ij = k;	jk = i;	ki = j
ji = -k;	kj = -i;	ik = -j
ii = jj = kk = -1;	ijk = -1;	

Le regole di moltiplicazione degli stessi vettori, nell'algebra degli ipercomplessi sono :

ij = k	jk = -i	ki = -j
ji = k	kj = -i	ik = -j
ii = jj = kk = -1	ijk = 1	

Per aumentare la velocità di calcolo quando viene usata l'algebra dei quaternioni si usa una stima della distanza. La prova del corretto funzionamento di questa stima della distanza, quando la formula è generalizzata da due a quattro dimensioni non c'è, ma la formula sembra comunque funzionare bene, nonostante la mancanza della prova matematica.
La presenza di una delle funzioni `sqr`, `cube` ecc., determina quale di queste funzioni viene usata

per f(h) nella formula ricorsiva h(n+1) = f(h(n)) + p. La maggior parte delle funzioni è valida solo se è presente la parola chiave `hypercomplex`. Con i quaternioni funzionano solamente `sqr` e `cube`. Le funzioni sono funzioni complesse estese alle quattro dimensioni.

Parola chiave della f	***Porta i valori 4D a :***
`sqr`	h*h
`cube`	h*h*h
`exp`	e elevato alla h
`reciprocal`	1/h
`sin`	seno di h
`asin`	arcoseno di h
`sinh`	seno iperbolico di h
`asinh`	inverso del seno iperbolico di h
`cos`	coseno di h
`acos`	arcocoseno di h
`cosh`	coseno iperbolico di h
`acosh`	inverso del coseno iperbolico di h
`tan`	tangente di h
`atan`	arcotangente di h
`tanh`	tangente iperbolica di h
`atanh`	inverso della tangente iperbolica di h
`log`	logaritmo naturale di h
`pwr(x,y)`	h elevato alla potenza complessa di x+iy

Un semplice esempio di frattale Julia è:

```
julia_fractal {
<-0.083,0.0,-0.83,-0.025>
quaternion
sqr
max_iteration 8
precision 15
}
```

Il primo rendering di frattali Julia usando i quaternioni è stato fatto da Alan Norton e più tardi da John Hart negli anni 80. Questa nuova possibilità di POV-Ray segue **Fractint** nell'idea di andare oltre a ciò che è noto nella letteratura matematica sull'uso degli ipercomplessi e generalizzando la formula ricorsiva per arrivare ad usare una quantità maggiore di funzioni oltre alla classica formula di Mandelbrot : z^2 + c. Con due dimensioni in più e diciotto funzioni con le quali lavorare, gli esploratori più intrepidi saranno in grado di creare nuove bestie frattali da mettere nell'iperspazio. Fatevi sotto !

7.5.2.7 Lathe

Gli oggetti torniti sono oggetti creati dalla rotazione di una curva bidimensionale attorno ad un asse. Questa curva è definita con un insieme di punti che sono connessi tra di loro con curve lineari, quadratiche e cubiche. La sintassi è :

```
lathe {
[ linear_spline | quadratic_spline | cubic_spline ]
NUMERO_DI_PUNTI,
<PUNTO_1>, <PUNTO_2>, ..., <PUNTO_n>
[ sturm ]
}
```

Il parametro `NUMERO_DI_PUNTI` specifica quanti punti bidimensionali formano la curva. Questi punti sono uniti fra di loro con curve lineari, quadratiche e cubiche come è specificato da una parola chiave facoltativa (quella di default è `linear_spline`). Dal momento che la curva non viene chiusa automaticamente, cioè il primo e l'ultimo punto non sono connessi automaticamente, è necessario che questo venga fatto a mano se vuoi una curva chiusa. La curva così definita è ruotata attorno all'asse y per formare l'oggetto che ha come centro l'origine.
Il seguente esempio crea un semplice oggetto che assomiglia ad un sottile cilindro, cioè un cilindro delimitato da una sottile superficie :

```
lathe {
linear_spline
5,
<2, 0>, <3, 0>, <3, 5>, <2, 5>, <2, 0>
pigment {Red}
}
```

Il cilindro ha il raggio interno di 2 e il raggio esterno di 3, avendo in tutto uno spessore di 1. La sua altezza è 5, è posto all'origine e sale verso l'alto, cioè l'asse di rotazione è l'asse y. Notate che il primo e l'ultimo punto sono uguali, in questo modo si genera una curva chiusa.
Le spline che sono usate dagli oggetti torniti e dai prismi sono un po' difficili da capire. Il concetto base delle spline è quello di disegnare una curva attraverso un insieme dato di punti in un certo modo. La spline lineare è la più semplice perché non consiste in altro nell'unire punti consecutivi con una linea. Questo significa che la curva è tracciata tra due punti solo in relazione alla loro reciproca posizione. Nessun'altra informazione è tenuta in conto. Le spline quadratiche e cubiche sono diverse per il loro funzionamento, non solo considerano anche la posizione di altri punti quando ne devono unire due, ma hanno anche un aspetto più smussato e nel caso delle spline cubiche producono un passaggio molto graduale da un punto all'altro.
Le spline quadratiche sono calcolate con funzioni quadratiche. Ciascuna curva unisce due punti consecutivi. Dal momento che questi due punti (chiamiamoli secondo e terzo) non sono sufficienti a descrivere una curva quadratica, è necessario specificarne un terzo, di cui si possa tenere conto per disegnare la curva. Matematicamente la relazione (la posizione reciproca sul piano bidimensionale) tra il primo ed il secondo punto determina la pendenza della curva al secondo punto. La pendenza della curva al terzo punto non è sotto controllo. Per questo le curve quadratiche appaiono molto più smussate delle curve lineari, ma il passaggio da un punto all'altro non è completamente smussato perché la pendenza dalle due parti del punto può essere diversa.
Le spline cubiche superano il problema della transizione da un punto all'altro che abbiamo ora descritto per le spline quadratiche. Infatti prendono in considerazione anche il quarto punto quando tracciano la curva tra il secondo ed il terzo. La pendenza al quarto punto è sotto controllo e ciò permette una transizione più smussata tra un punto ed un altro.
Devi notare che il numero dei segmenti che compongono la curva dipende dal tipo di spline usato.

Per le spline lineari si avranno n-1 segmenti che connettono i punti P[i], i=1,...,n. Una spline quadratica darà n-2 segmenti perché l'ultimo punto è usato solamente per determinare la pendenza (quindi sono necessari come minimo tre punti per definire una spline quadratica). Lo stesso vale per le spline cubiche, dove si avranno n-3 segmenti, con il primo e l'ultimo punto usati solo per il calcolo della pendenza (quindi servono almeno quattro punti).
Se vuoi ottenere una spline quadratica o cubica chiusa con transizioni smussate da un punto ad un altro, è necessario che ti assicuri che, nel caso delle spline cubiche , P[n-1] = P[2] (per ottenere una curva chiusa), P[n] = P[3] e P[n-2] = P[1] (per ottenere transizioni graduali). Nel caso di una spline quadratica deve valere che P[n-1] = P[1] (per avere una curva chiusa) e P[n] = P[2].
Con le spline quadratiche è possibile usare l'algoritmo di Sturm, più lento, ma più accurato. Dal momento che è necessario risolvere un polinomio di secondo grado, l'algoritmo di Sturm non è necessario per le spline lineari. Nel caso di spline cubiche invece viene spesso usato perché si deve risolvere un polinomio di sesto grado.

7.5.2.8 Prisma

Il prisma è un oggetto generato dalla traslazione di una o più curve chiuse ed estese su due dimensioni lungo un asse. Queste curve sono definite da un insieme di punti che vengono collegati insieme da spline lineari, quadratiche o cubiche.

La sintassi del prisma è :

```
prism {
[ linear_sweep | conic_sweep ]
[ linear_spline | quadratic_spline | cubic_spline ]
ALTEZZA1,
ALTEZZA2,
NUMERO_TOTALE_DI_PUNTI,
<PUNTO_1>, <PUNTO_2>, ..., <PUNTO_n>
[ open ]
[ sturm ]
}
```

L'oggetto *prisma* ti permette di inserire un qualunque numero di frasi che definiscono *sotto - prismi* all'interno di una frase `prism{...}` (purché usino gli stessi tipi di spline e di estrusione). Ovunque si sovrappone un numero pari di sotto - prismi, appare un 'buco'.

La sintassi che definisce il prisma dipende dal tipo di spline usato. Questa è la sintassi del *prisma con spline lineare* :

```
prism {
linear_spline
ALTEZZA1,
ALTEZZA2,
NUMERO_TOTALE_DI_PUNTI,
<A_1>, <A_2>, ..., <A_na>, <A_1>,
<B_1>, <B_2>, ..., <B_nb>, <B_1>,
<C_1>, <C_2>, ..., <C_nc>, <C_1>,
...
}
```

Ciascuno dei sotto - prismi deve essere *chiuso* ripetendo il primo punto al termine della sequenza.

Se ciò non avviene, POV-Ray visualizza un messaggio di avvertimento ed il prisma non viene calcolato (ma con le spline lineari, viene chiuso automaticamente ed il calcolo prosegue). Ciò implica che tutti i punti di un prisma devono essere diversi gli uni dagli altri, tranne naturalmente il primo e l'ultimo che devono essere identici.. Ciò si applica a tutti i tipi di spline, sebbene i *punti di controllo* delle spline quadratiche e cubiche possano essere scelti arbitrariamente.

L'ultimo sotto - prisma di un prisma a spline quadratica è chiuso automaticamente, come l'ultimo sotto - poligono contenuto nei poligoni, se il primo e l'ultimo punto non coincidono. Ciò rende molto semplice la conversione tra poligoni e prismi. Le spline quadratiche e cubiche non sono mai chiuse automaticamente.

La sintassi per un *prisma a spline quadratica* è la seguente :

```
prism {
quadratic_spline
ALTEZZA1,
ALTEZZA2,
NUMERO_TOTALE_DI_PUNTI,
<CL_A>, <A_1>, <A_2>, ..., <A_na>, <A_1>,
<CL_B>, <B_1>, <B_2>, ..., <B_nb>, <B_1>,
<CL_C>, <C_1>, <C_2>, ..., <C_nc>, <C_1>,
...
}
```

I sotto - prismi a spline quadratica hanno bisogno di un punto di controllo aggiuntivo all'inizio di ogni sequenza di punti, per determinare la direzione della curva al suo inizio.

Infine, la sintassi per un *prisma a spline cubica*.

```
prism {
cubic_spline
ALTEZZA1,
ALTEZZA2,
NUMERO_TOTALE_DI_PUNTI,
<CL_A1>, <A_1>, <A_2>, ..., <A_na>, <A_1>, <CL_A2>,
<CL_B1>, <B_1>, <B_2>, ..., <B_nb>, <B_1>, <CL_B2>,
<CL_C1>, <C_1>, <C_2>, ..., <C_nc>, <C_1>, <CL_C2>,
...
}
```

In aggiunta alla serie di punti (chiusa, nel senso che il primo e l'ultimo devono coincidere), ogni sotto - prisma deve avere due punti aggiuntivi, uno all'inizio ed uno alla fine della sequenza, per determinare la direzione della spline all'inizio ed alla fine.

Il parametro `NUMERO_TOTALE_DI_PUNTI` determina quanti punti (a due dimensioni, giacenti nel piano x-z) formano le curve, *compresi i punti di controllo necessari per le spline quadratiche e cubiche*. Le curve vengono poi traslate lungo l'asse delle y da `ALTEZZA1` ad `ALTEZZA2`, per formare il prisma. Per default viene usata una traslazione lineare. Le pareti del prisma sono cioè perpendicolari al piano x-z e la dimensione della curva non cambia durante la traslazione.
Si può anche usare la traslazione conica (specificata dalla parola chiave `conic_sweep`) che dà un prisma con pareti simili a quelle di un cono, rimpicciolendo la curva nel corso della traslazione.

Come accade per i cilindri, il prisma è normalmente chiuso alle estremità. Si possono aprire le estremità usando la parola chiave `open`. In questo caso, si dovrebbe evitare di utilizzare il prisma in oggetti CSG in quanto il risultato potrebbe non corrispondere alle aspettative.

L'esempio seguente crea un semplice prisma che assomiglia ad una fetta di torta :

```
prism {
linear_sweep
linear_spline
0, 1,
4,
<-1, 0>, <1, 0>, <0, 5>, <-1, 0>
pigment {Red}
}
```

Per una dettagliata spiegazione del concetto di spline, leggi la descrizione dell'oggetto 'lathe' (vedi il paragrafo "Oggetti Torniti").

Il metodo di risoluzione di Sturm, più lento ma più accurato, può essere utilizzato *solo* con i prismi a spline cubica, nel caso in cui l'oggetto non venga renderizzato correttamente. I prismi a spline lineare e quadratica non hanno bisogno di questo metodo.

7.5.2.9 Sfera

La sintassi dell'oggetto *sfera* è la seguente :

```
sphere {
<CENTRO>, RAGGIO
}
```

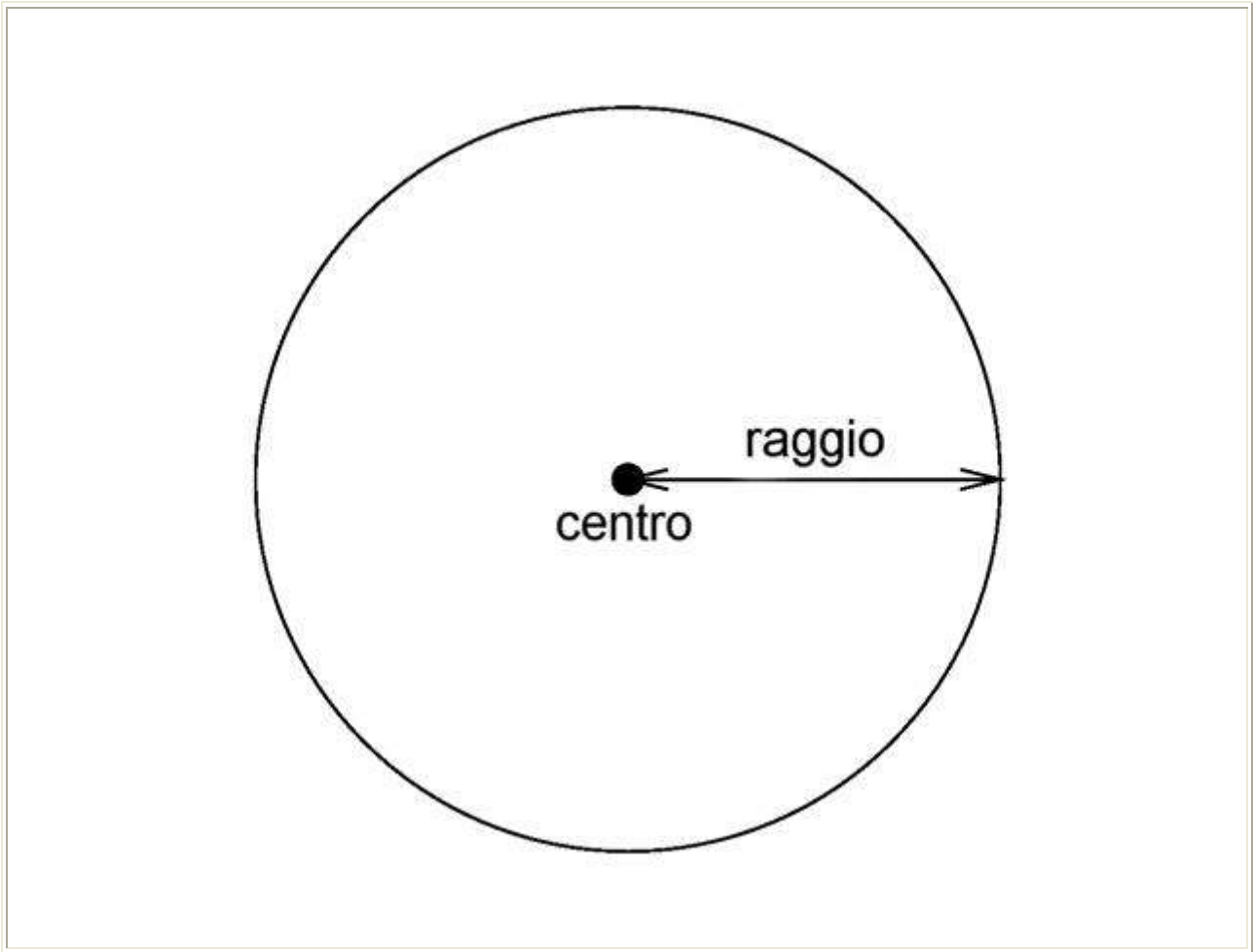



Figura 203- La geometria della sfera

dove `<CENTRO>` è un vettore che specifica le coordinate x, y, z del centro della sfera e `RAGGIO` è un valore decimale che specifica il raggio. Le sfere possono essere ridimensionate in maniera non uniforme lungo i tre assi, dando oggetti ellissoidali.

Dato che le sfere sono oggetti molto ottimizzati, possono costituire ottimi oggetti delimitanti, se il bounding manuale è necessario.

7.5.2.10 Superellissoidi

L'ellissoide superquadrico, o *superellissoide* è un'estensione dell'ellissoide quadrico. Può essere usato per creare parallelepipedi e cilindri con gli spigoli smussati ed altri oggetti interessanti. Matematicamente è dato dall'equazione :

```
f(x, y, z) = (|x|^(2/e) + |y|^(2/e)) ^ (e/n) + |z|^(2/n) - 1 = 0
```

I valori di *e* ed *n*, chiamati esponenti *est* - *ovest* e *nord* - *sud*, rispettivamente, determinano la forma del superellissoide. Entrambi devono essere maggiori di zero. Ad esempio, la sfera è data da *e*=1 ed *n*=1.
La sintassi del superellissoide, che per default è posizionato all'origine, è la seguente :

```
superellipsoid { <e, n> }
```

Due oggetti utili sono il 'parallelepipedo arrotondato' ed il 'cilindro arrotondato'. Vengono dichiarati

nel seguente modo :

```
#declare Parallelepipedo_Arrot = superellipsoid { <r, r> }
#declare Cilindro_Arrot = superellipsoid { <1, r> }
```

Il valore di *r* determina quanto gli spigoli sono smussati e deve essere compreso tra 0 ed 1. Minore è il valore di *r*, più netti saranno gli spigoli.

Valori molto piccoli di *e* ed *n* potrebbero causare problemi nel calcolo (il metodo di Sturm non può essere usato).

7.5.2.11 Superfici di Rotazione

Le superfici di rotazione (SOR, ovvero *surface of revolution*) sono generate ruotando una curva attorno all'asse delle y. La curva è composta di punti che descrivono la dipendenza del raggio dall'altezza sull'asse di rotazione. La sintassi dell'oggetto SOR è :

```
sor {
NUMERO_DI_PUNTI,
<PUNTO0>, <PUNTO1>, ..., <PUNTOn-1>
[ open ]
[ sturm ]
}
```

I punti da `<PUNTO0>` a `<PUNTOn-1>` sono vettori a due dimensioni composti dal raggio e dall'altezza corrispondente, cioè la posizione sull'asse di rotazione. Questi punti sono collegati insieme da una curva e ruotati attorno all'asse delle y, a formare l'oggetto SOR. Il primo e l'ultimo punto sono utilizzati come punti di controllo della curva. La funzione utilizzata per calcolare la curva che congiunge i punti è simile alla funzione spline utilizzata per gli oggetti lathe. La differenza è che gli oggetti SOR sono meno flessibili, poiché devono sottostare alle leggi che regolano tutte le funzioni matematiche e cioè che ad ogni punto dell'asse y deve corrispondere al massimo un punto della funzione, cioè un valore di raggio. Non si possono usare curve chiuse per un oggetto SOR.

La parola chiave `open` ti permette di rimuovere le chiusure alle estremità dell'oggetto. Se viene utilizzata, non si dovrebbe usare l'oggetto SOR in oggetti CSG, poiché potrebbe dare risultati errati.

L'oggetto SOR è utile per creare bottiglie, vasi ed oggetti simili. Un semplice vaso potrebbe essere :

```
#declare Vaso = sor {
7,
<0.000000, 0.000000>
<0.118143, 0.000000>
<0.620253, 0.540084>
<0.210970, 0.827004>
<0.194093, 0.962025>
<0.286920, 1.000000>
<0.468354, 1.033755>
open
}
```

Ci si potrebbe chiedere perché ci sia bisogno di un oggetto SOR quando si ha già a disposizione un

oggetto 'lathe', molto più flessibile. Il motivo è molto semplice. I test di intersezione con un oggetto SOR richiedono di risolvere un'equazione di terzo grado, mentre un oggetto lathe richiede la risoluzione di un'equazione di sesto (e si ha bisogno di una spline cubica per ottenere la stessa gradualità nella curva). Dato che la maggior parte degli oggetti SOR e lathe sono costituiti da parecchi punti, ciò comporterà una grossa differenza nel tempo di rendering. Inoltre, le soluzioni dell'equazione di terzo grado sono più accurate.
Se l'oggetto non viene calcolato correttamente, si può utilizzare il metodo di Sturm, più lento, ma più accurato.

Le seguenti spiegazioni sono per il lettore interessato all'aspetto matematico dell'oggetto SOR. Sebbene non sia necessario leggerle, possono essere d'aiuto per capire come esso viene calcolato.

La funzione che viene ruotata attorno all'asse delle y per ottenere l'oggetto SOR è :

```
r^2 = f(h) = A*h^3 + B*h^2 + C*h + D
```

Dove *r* è il raggio e *h* è l'altezza. Dato che questa è una funzione cubica in *h*, è sufficientemente versatile e permette di ottenere curve regolari.

La curva stessa è definita da un insieme di *n* punti P(*i*), con *i* che varia tra 0 ed *n*-1, che vengono interpolati utilizzando una funzione per ogni segmento della curva. Un segmento *j* con *j* che varia tra 1 ed *n*-3 va dal punto P(*j*) al P(*j*+1) ed utilizza i punti P(*j*-1) e P(*j*+2) per determinare la pendenza della curva alle estremità. Se ci sono *n* punti, otterremo *n*-3 segmenti. Ciò significa che sono necessari almeno quattro punti per ottenere una curva. I coefficienti A(*j*), B(*j*), C(*j*), D(*j*) sono calcolati per ogni segmento per mezzo delle equazioni :

$$b = M \cdot x$$

$$b = \begin{pmatrix} r(j)^2 \\ r(j+1)^2 \\ 2 \cdot r(j) \cdot (r(j+1) - r(j-1)) / (j(j+1) - h(j-1)) \\ 2 \cdot r(j+1) \cdot (r(j+2) - r(j)) / (h(j+2) - h(j)) \end{pmatrix}$$

$$m = \begin{pmatrix} h(j)^3 & h(j)^2 & h(j) & 1 \\ h(j+1)^3 & h(j+1)^2 & h(j+1) & 1 \\ 3 \cdot h(j)^2 & 2 \cdot h(j) & 1 & 0 \\ 3 \cdot h(j+1)^2 & 2 \cdot h(j+1) & 1 & 0 \end{pmatrix}$$

$$x = \begin{pmatrix} A(j) \\ B(j) \\ C(j) \\ D(j) \end{pmatrix}$$

dove r(*j*) è il raggio ed h(*j*) è l'altezza del punto P(*j*).
La figura sotto mostra la configurazione dei punti P(*i*), la posizione del segmento j, e la curva definita da questo segmento.

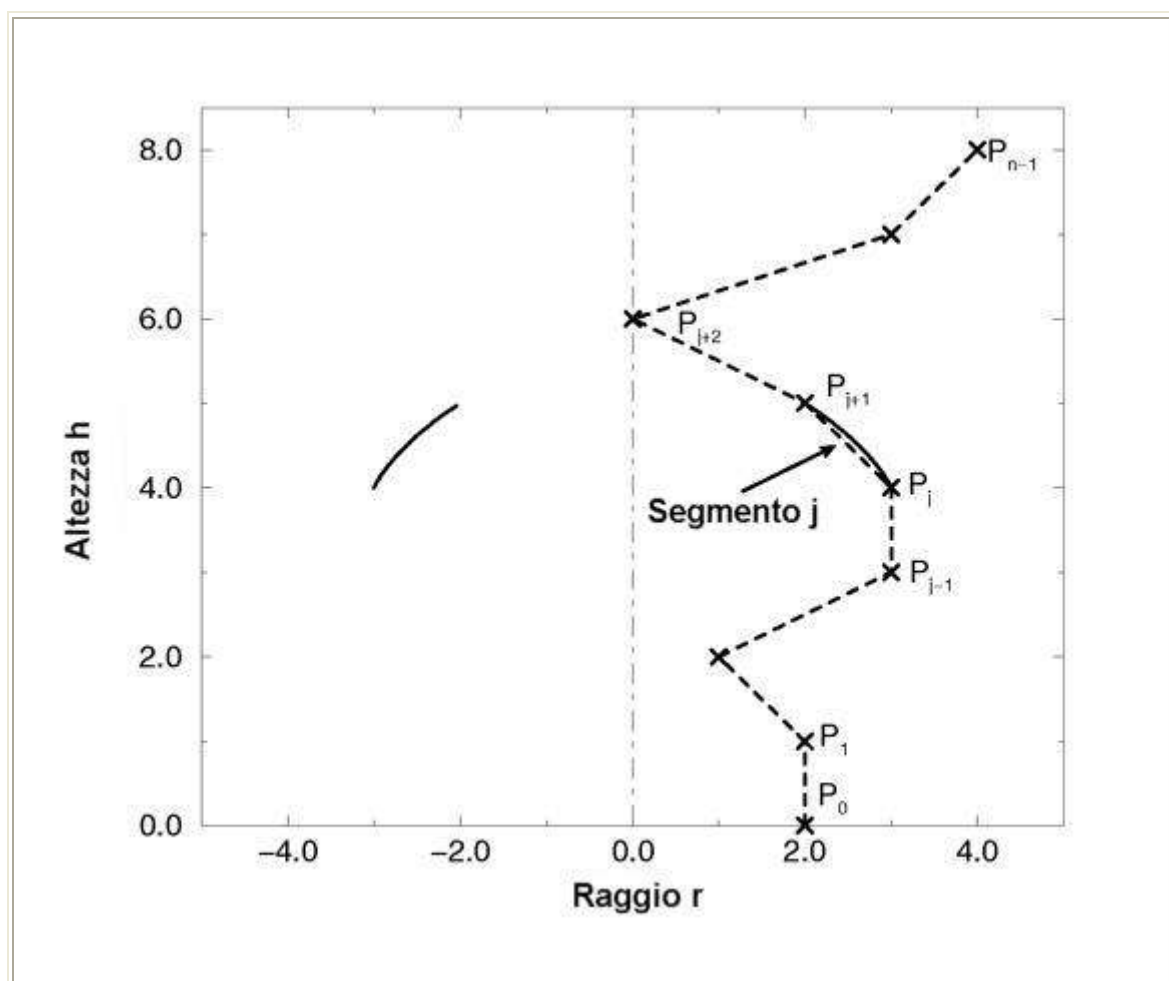



Fig. 204 - J-esimo segmento di un gruppo di n-3 segmenti in una configurazione composta da n punti. Il punto descrive la curva di una superficie dii rotazione.

7.5.2.12 Testo

L'oggetto *testo* permette di creare del testo tridimensionale. Al momento è possibile usare solamente caratteri *True Type*™, ma l'impostazione della sintassi permette che in futuro possano essere aggiunti altri tipi. La sintassi è :

```
text {
ttf "NOMEFONT.TTF",
```

```
"STRINGA_DI_TESTO",
SPESSORE, VETTORE_OFFSET
}
```

Dove **nomefont.ttf** indica il nome del file di caratteri true type. E' una stringa composta da lettere o espressioni. L'espressione che segue è il testo che dovrà essere visualizzato e anch'esso può essere costituito da una stringa composta da lettere o espressioni. Vedere il paragrafo "Stringhe".
Il testo sarà posizionato in modo che l'angolo in basso a sinistra si trovi nell'origine e si estenderà lungo la direzione positiva dell'asse x. La base della scritta segue l'asse x. La superficie superiore del carattere giace sul piano x-y e il testo è estruso nella direzione positiva dell'asse z. Lo spessore è specificato dal numero decimale `SPESSORE`.
I caratteri hanno in generale una dimensione per la quale è sufficiente 1 unità di spazio verticale. L'altezza dei caratteri è circa di 0.5, 0.75 unità.
La spaziatura orizzontale è calcolata da POV-Ray grazie alle informazioni contenute nel carattere. Il vettore `VETTORE_OFFSET`, obbligatorio, definisce ogni ulteriore spaziatura tra i caratteri. Normalmente il valore che attribuirai a questo vettore sarà 0. Specificando il valore 0.1*x, sarà possibile ottenere una spaziatura aggiuntiva tra i caratteri. Gli oggetti testo possono utilizzare solo caratteri stampabili. Non si possono visualizzare caratteri quali 'return' (a capo), 'avanzamento linea' (*line feed*, LF), tabulature e backspace.

7.5.2.13 Toro

Il toro è rappresentato da una polinomiale di 4° ordine che assomiglia ad una ciambella. Dal momento che quest'oggetto è utilissimo e le quartiche sono molto difficili da definire, POV-Ray permette di prendere una scorciatoia e definire il toro così :

```
torus {
MAGGIORE, MINORE
[ sturm ]
}
```

dove `MAGGIORE` è un valore decimale da attribuire al raggio maggiore e `MINORE` è un decimale che imposta il raggio minore. Il raggio maggiore parte dal centro del toro e arriva fino alla metà dell'anello, mentre il raggio minore parte da questo e arriva fino al bordo del toro. Il toro è centrato nell'origine e giace nel piano x-z , attraversato dall'asse y.

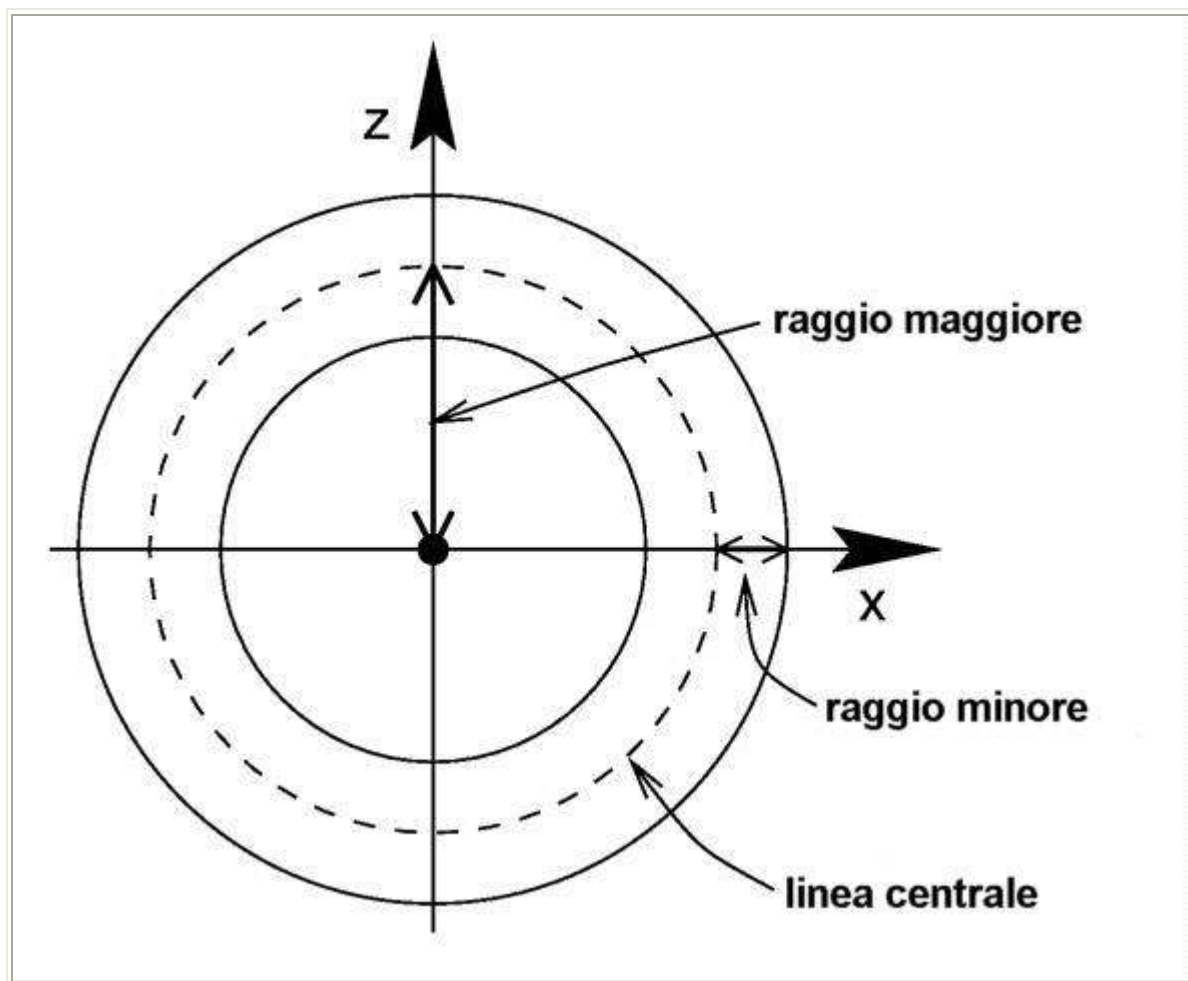



Figura 205-Raggi maggiore e minore del toro.

Il toro è delimitato (bounded) da due cilindri e due anelli che formano un finissimo cilindro. Grazie a questo cilindro l'oggetto *toro* è notevolmente ottimizzato rispetto alla sua definizione come quartica. I test di intersezione del toro, che richiedono la soluzione di una polinomiale di 4° ordine, vengono calcolati solo quando il cilindro viene colpito. In questo modo si è riusciti ad evitare molti lunghi calcoli.
I calcoli per le polinomiali di ordine superiore devono invece essere molto accurati. Se il toro appare in modo scorretto è possibile aggiungere la parola chiave `sturm` dopo il valore del raggio minore, per fare usare a POV-Ray l'algoritmo di Sturm, più accurato, ma più lento.

7.5.3 Superfici Primitive Finite

Ci sono sei oggetti di dimensioni finite ma privi di spessore : superfici bicubiche, dischi, triangoli, triangoli smussati, poligoni e mesh. Possono essere combinati con unioni CSG (o all'interno di una frase `clipped_by{...}`. Dal momento che questi oggetti sono finiti, POV-Ray può usare il bounding automatico per aumentare la velocità di rendering.. Come tutti gli altri oggetti possono essere spostati, ruotati e ridimensionati.

7.5.3.1 Superfici Bicubiche

Una superficie bicubica è una superficie curva tridimensionale simulata da un insieme di triangoli. POV-Ray supporta un tipo di superficie bicubica chiamata *superficie di Bezier*. Una superficie bicubica è definita come segue :

```
bicubic_patch {
```

```
type TIPO_DI_SUPERFICIE
flatness BIDIMENSIONALITA'
u_steps NUMERO_DI_U_STEPS
v_steps NUMERO_DI _V_STEPS
<CP1>, <CP2>, <CP3>, <CP4>,
<CP5>, <CP6>, <CP7>, <CP8>,
<CP9>, <CP10>, <CP11>, <CP12>,
<CP13>, <CP14>, <CP15>, <CP16>
}
```

La parola chiave `type` è seguita da un numero che può essere 0 oppure 1.Per il tipo 0 solo i punti di controllo sono memorizzati da POV-Ray. Questo significa che è necessaria una piccola quantità di memoria, ma POV-Ray dovrà fare molti calcoli al momento del rendering. Il tipo 1 suddivide la superficie in molte 'sotto - superfici', incrementando notevolmente la velocità del rendering, ma aumentando la quantità di memoria necessaria.
I quattro parametri, `type`, `flatness`, `u_steps` e `v_steps` possono apparire in qualunque ordine. Sono seguiti da sedici vettori che definiscono le coordinate x, y, z dei sedici punti di controllo che definiscono la superficie. La superficie tocca i quattro punti d'angolo <CP1>, <CP4>, < CP13> e <CP16>, mentre gli altri dodici punti danno la forma alla superficie (si immagini di montare una tenda da campeggio : alla base, il telo coincide con i picchetti, mentre gli altri tiranti *non sono* sulla superficie del telo).
La superficie è racchiusa nell'intelaiatura convessa formata dai sedici punti di controllo. Le parole chiave `u_steps` e `v_steps` sono seguite da un valore decimale che specifica quante righe e quante colonne di triangoli devono essere usate come minimo per creare la superficie. Il numero massimo di singole parti della superficie che vengono calcolate da POV-Ray si può ricavare dalla seguente formula :

```
sotto-superfici = 2^u_steps * 2^v_steps
```

Questo significa che i valori di `u_steps` e `v_steps` dovrebbero essere mantenuti al di sotto di 4. Molte superfici risultano buone con un valore 3 assegnato ai due parametri, che crea 64 sotto - superfici (per un totale di 128 triangoli smussati).
Quando POV-Ray calcola le superfici di Bezier, esegue un test su ogni sotto - superficie per vedere se essa è abbastanza piatta da potersi approssimare ad un rettangolo. La parola chiave che controlla questo test è `flatness`. Valori tipici per questo parametro sono compresi tra 0 ed 1 (più basso è il valore, più lento sarà il calcolo).
Se il valore di `flatness` è zero, POV-Ray suddivide comunque la superficie come specificato da `u_steps` e `v_steps`. Se `flatness` è maggiore di 0, allora ogni volta che la superficie è divisa, POV-Ray controlla se c'è bisogno di eseguire un'ulteriore suddivisione.
Usare valori diversi da zero per `flatness` comporta sia vantaggi che svantaggi. I vantaggi sono :

- se la superficie non è molto curva, allora POV-Ray se ne accorgerà e non perderà tempo cercando di calcolare parti non influenti.
- se la superficie è molto curva in zone molto ristrette, POV-Ray si concentrerà su queste, creando ulteriori suddivisioni solo dove sono necessarie.

Il maggiore svantaggio è invece che POV-Ray termina la suddivisione a livelli diversi nelle diverse zone della superficie, è questo può far sì che la superficie appaia *spezzata*. Questo fenomeno si manifesta in zone trasparenti della superficie. La gravità di questo inconveniente dipende più che altro dall'angolo dal quale si osserva l'oggetto.
Come i triangoli, le superfici bicubiche non sono state ideate per essere scritte a mano. Questi

oggetti dovrebbero essere creati da programmi esterni, che sono generalmente disponibili alla stessa fonte dalla quale si è ottenuto POV-Ray.

```
bicubic_patch {
type 1
flatness 0.01
u_steps 4
v_steps 4
<0, 0, 2>, <1, 0, 0>, <2, 0, 0>, <3, 0,-2>,
<0, 1 0>, <1, 1, 0>, <2, 1, 0>, <3, 1, 0>,
<0, 2, 0>, <1, 2, 0>, <2, 2, 0>, <3, 2, 0>,
<0, 3, 2>, <1, 3, 0>, <2, 3, 0>, <3, 3, -2>
}
```

I triangoli in una superficie bicubica sono automaticamente smussati usando la normale interpolazione, ma è compito di chi le crea (o del programma che le crea) creare punti di controllo che uniscano con uniformità più superfici.

7.5.3.2 Disco

Un altro oggetto piatto di dimensioni finite realizzabile con POV-Ray è il *disco*. Questo oggetto è molto sottile e non ha spessore. Se vuoi un disco con almeno un po'di spessore dovrai usare un cilindro molto basso. Il *disco* è definito così :

```
disc {
<CENTRO>, <NORMALE>, RAGGIO [, RAGGIO_DEL_BUCO ]
}
```

Il vettore <CENTRO> definisce le coordinate x, y, z del centro del disco. Il vettore <NORMALE> indica la sua orientazione descrivendo il vettore normale alla superficie. E' seguito da un decimale che specifica il raggio. E' facoltativo aggiungere un altro decimale che specifica il raggio di un buco da ritagliarsi all'interno del disco, a partire dal centro.

7.5.3.3 Mesh

L'oggetto *mesh* può essere usato per memorizzare in modo efficiente molti triangoli. La sintassi è :

```
mesh {
triangle {
<VERTICE1>, <VERTICE2>, <VERTICE3>
[ texture { ... } ]
}
smooth_triangle {
<VERTICE1>, <NORMALE1>,
<VERTICE2>, <NORMALE2>,
<VERTICE3>, <NORMALE3>
[ texture { ... } ]
}
[ hierarchy ON - OFF ]
}
```

Un qualunque numero di triangoli, smussati o no, può essere usato e ad ogni triangolo può essere assegnata una singola texture. La texture deve essere dichiarata *prima* dei triangoli. Non è possibile

usare frasi `texture{...}` all'interno della definizione dei triangoli. Questa è una restrizione necessaria per una memorizzazione efficiente delle texture.
I componenti della mesh sono delimitati internamente da una gerarchia di parallelepipedi, per accelerare i test di intersezione raggio - oggetto. La gerarchia di delimitazione può essere disattivata mediante l'uso della parola chiave `hierarchy` seguita da `on` oppure `off`. Si dovrebbe disattivarla solo se si ha poca memoria, o se l'oggetto consiste di un numero molto piccolo di triangoli.
Copie di un oggetto mesh si basano sugli stessi dati per i triangoli e quindi consumano poca memori in più. Si possono renderizzare velocemente centinaia di copie di un oggetto di 10000 facce, senza correre il rischio di esaurire la memoria (sempre che l'oggetto originale possa essere memorizzato).
L'oggetto *mesh* ha due vantaggi rispetto ad una semplice unione di triangoli : richiede meno memoria e viene trasformato più velocemente. Le richieste di memoria sono ridotte memorizzando in maniera efficiente i vertici e le normali dei triangoli. Il tempo di analisi (*parsing*) per gli oggetti *mesh* trasformati è ridotto perché viene trasformato l'intero oggetto e non i singoli triangoli, come invece accade per le unioni.
L'oggetto *mesh* può includere (per adesso) solo triangoli e triangoli smussati. Questa restrizione è suscettibile di futuri cambiamenti, permettendo l'inserimento di componenti poligonali.

7.5.3.4 Poligoni

I poligoni sono utili per creare quadrati, rettangoli e tutte i poligoni piani con un numero di angoli superiore a tre. La sintassi è :

```
polygon {
NUMERO_TOTALE_DI_PUNTI,
<A_1>, <A_2>, ..., <A_na>, <A_1>,
<B_1>, <B_2>, ..., <B_nb>, <B_1>,
<C_1>, <C_2>, ..., <C_nc>, <C_1>,
...
}
```

I punti da <A_1> ad <A_na> descrivono il primo 'sotto - poligono', i punti da <B_1> a <B-nb> descrivono il secondo e così via. Un poligono può contenere un qualunque numero di sotto - poligoni, sovrapposti o no. Nelle zone dove si sovrappone un numero pari di poligoni, l'oggetto appare 'bucato'. Tutto ciò di cui ci si deve assicurare è che tutti questi poligoni siano chiusi, ripetendo il primo punto di ogni sotto - poligono alla fine della sequenza di punti che lo definisce. Ciò implica che tutti i punti di un sotto - poligono devono essere diversi.
Se l'ultimo sotto - poligono non è chiuso, viene visualizzato un messaggio di avvertimento e POV-Ray lo chiude automaticamente. Ciò è utile in quanto i poligoni importati da un altro programma possono non essere chiusi, cioè il loro primo ed ultimo punto possono non essere uguali.
Tutti i punti di un poligono sono descritti di vettori a tre dimensioni che ne indicano la posizione e che *devono giacere su un piano*. Se ciò non accade, viene visualizzato un programma di errore e POV-Ray si ferma. Se vengono usati vettori a due dimensioni, POV-Ray assume che la terza coordinata sia uguale a zero.

Un poligono quadrato che si adatta perfettamente alla mappa di immagine quadrata di default, è :

```
polygon {
4,
<0, 0>, <0, 1>, <1, 1>, <1, 0>
texture {
finish { ambient 1 diffuse 0 }
pigment { image_map { gif "prova.gif" } }
```

```
}
//scalare e ruotare come serve
}
```

La funzione 'sotto - poligono' può essere usata per creare oggetti complessi come la lettera 'P', dove si crea un foro in un altro poligono :

```
#declare P = polygon {
12,
<0, 0>, <0, 6>, <4, 6>, <4, 3>, <1, 3>, <1, 0>, <0, 0>,
<1, 4>, <1, 5>, <3, 5>, <3, 4>, <1, 4>
}
```

Il primo 'sotto - poligono' (prima riga) descrive la sagoma esterna della lettera. Il secondo 'sotto - poligono' (seconda riga) descrive il foro rettangolare che viene praticato nella parte superiore della lettera 'P'. Entrambi i poligoni sono chiusi, cioè, i punti iniziali e finali sono gli stessi.

La possibilità di creare dei fori all'interno di un poligono è data dai test di intersezione interno - esterno che POV-Ray effettua. Un punto si considera essere all'interno di un poligono se una linea retta tracciata da questo punto in una direzione arbitraria attraversa un contorno un numero dispari di volte (questo è conosciuto come *Teorema delle Curve di Jordan*).

Un altro esempio molto complesso, che mostra un grande triangolo con tre piccoli fori e tre triangolini separati, è dato sotto :

```
polygon {
28,
<0, 0> <1, 0> <0, 1> <0, 0> // triangolo grande esterno
<.3, .7> <.4, .7> <.3, .8> <.3, .7> // triangolino esterno#1
<.5, .5> <.6, .5> <.5, .6> <.5, .5> // triangolino esterno#2
<.7, .3> <.8, .3> <.7, .4> <.7, .3> // triangolino esterno #3
<.5, .2> <.6, .2> <.5, .3> <.5, .2> // triangolo interno#1
<.2, .5> <.3, .5> <.2, .6> <.2, .5> // triangolo interno #2
<.1, .1> <.2, .1> <.1, .2> <.1, .1> // triangolo interno#3
}
```

7.5.3.5 Triangoli e Triangoli Smussati

La primitiva *triangolo* è disponibile per ottenere oggetti più complessi di quelli che si possono ottenere dalle normali primitive. I triangoli non sono solitamente creati a mano ma vengono convertiti da altri file o generati da altri programmi. Un triangolo è definito dalla seguente sintassi :

```
triangle {
<VERTICE1>, <VERTICE2>, <VERTICE3>
}
```

dove <VERTICEn> è un vettore definito dalle coordinate x, y, z di ciascun vetrice del triangolo. Dal momento che i triangoli sono superfici perfettamente piane, sarà necessario un numero molto grande di triangoli per approssimare una superficie curva. Comunque, la nostra capacità di percezione delle superfici piane dipende principalmente dal modo in cui sono definite le luci e le ombre. Modificando artificialmente le normali alla superficie, possiamo simulare superfici lisce ed uniformi e nascondere così gli spigoli tra i triangoli adiacenti. La primitiva `smooth_triangle` è

usata proprio per questo scopo. I triangoli smussati usano un formula chiamata *Interpolazione delle Normali di Phong* per calcolare la normale alla superficie per ogni punto del triangolo, basandosi sui vettori normali che vengono definiti per i tre vertici. Questo procedimento permette di fare apparire i triangoli come superfici curve. Un triangolo smussato è definito così :

```
smooth_triangle {
<VERTICE1>, <NORMALE1>,
<VERTICE2>, <NORMALE2>,
<VERTICE3>, <NORMALE3>
}
```

dove i vertici sono definiti come in un normale triangolo e `<NORMALE1>` è un vettore che descrive la direzione della normale alla superficie per ogni vertice. Questi vettori sono veramente difficoltosi da calcolare a mano, ma in ogni caso i triangoli smussati sono quasi sempre generati da programmi esterni. Per arrivare a risultati uniformi, ogni triangolo che condivide un vertice con un altro dovrebbe avere o stesso vettore normale relativo a quello stesso vertice. Generalmente, i vettori normali che servono a questo scopo dovrebbero essere la media di tutti i vettori normali dei triangoli che condividono quel punto.

7.5.4 Primitive Solide Infinite

Ci sono cinque oggetti polinomiali primitivi che sono infiniti e non rispondono al bounding automatico : piano, cubiche, polinomiali, quadriche e quartiche. Possono essere combinati con unioni CSG (o all'interno di una frase `clipped_by{...}`. Come tutti gli oggetti possono essere spostati, ridimensionati e ruotati.

7.5.4.1 Piano

Il piano è un modo molto semplice per definire una superficie infinita. Il piano è definito come segue :

```
plane { <NORMALE>, DISTANZA }
```

Il vettore `<NORMALE>` definisce la normale alla superficie. Una normale alla superficie è un vettore perpendicolare ad essa. Questo vettore è seguito da un valore decimale che definisce la distanza dall'origine lungo il vettore normale (questo è valido solamente se il vettore normale ha lunghezza unitaria, vedremo perché). Per esempio :

```
plane { <0, 1, 0>, 4 }
```

Questa frase definisce un piano che ha la superficie superiore orientata lungo la direzione positiva dell'asse y ed è distante quattro unità dall'origine. Dal momento che molti piani sono definiti da un vettore normale che va nella direzione di un asse, spesso vedrai quegli stessi piani definiti utilizzando gli identificatori x, `y` e `z`. L'esempio chiarisce il concetto :

```
plane { y, 4 }
```

Il piano si estende all'infinito nella direzione dell'asse x e dell'asse z.. In concreto, divide il "mondo" in due parti. Per definizione il vettore normale punta verso l'esterno del piano mentre ogni punto dall'altra parte dal vettore è definito come interno. Questa distinzione tra interno ed esterno è importante solo quando si usano piani negli oggetti CSG e in frasi `clipped_by{...}`.
Un piano è chiamato polinomiale perché è definito da un polinomio di 1° grado. Dato un piano :

```
plane { <A, B, C>, D }
```

questo può essere rappresentato dall'equazione :

```
A*x + B*y + C*z - D*sqrt(A^2 + B^2 + C^2) = 0.
```

Quindi il nostro piano di esempio, `plane { y,4 }` è in effetti rappresentato dall'equazione polinomiale y=4. Si può immaginarla come l'insieme di tutti i punti che la coordinata y uguale a 4 indipendentemente dai valori di x e z.
Questa equazione è detta polinomiale del primo ordine, poiché ogni termine contiene solo potenze di primo grado di x, y, z. Un'equazione di secondo grado contiene termini come x^2, y^2, z^2, xy, xz e yz. Un altro nome per le equazioni di secondo grado è *quadriche*. Le polinomiali di terzo ordine sono dette *cubiche* e quelle di quarto sono dette *quartiche*. Questi oggetti sono descritti nei paragrafi seguenti.

7.5.4.2 Polinomiale, Cubica e Quartica

Superfici polinomiali di ordine maggiore possono essere definite usando una *polinomiale* (parola chiave `poly`).
La sintassi è :

```
poly { ORDINE, <T1, T2, T3, .... Tm> }
```

Dove `ORDINE` è un numero interno compreso tra 2 e sette che specifica l'ordine dell'equazione e `T1, T2, ... Tm` sono numeri decimali che specificano i coefficienti dell'equazione. Ci sono *m* termini di questo tipo dove

```
m = ((ORDINE+1)*(ORDINE+2)*(ORDINE+3))/6.
```

Un metodo alternativo per specificare polinomiali di terzo grado è :

```
cubic { <T1, T2,... T20> }
```

Anche le polinomiali di quarto grado possono essere specificate con :

```
quartic { <T1, T2,... T35> }
```

Qui di seguito diamo una descrizione più matematica delle quartiche per coloro che sono interessati. Le superfici quartiche sono superfici di 4° ordine che possono essere usate per descrivere un ampio insieme di oggetti, inclusi i tori. L'equazione generale di una quartica in tre variabili è (tenetevi forte) :

```
a00 x^4 + a01 x^3 y + a02 x^3 z+ a03 x^3 + a04 x^2 y^2+
a05 x^2 y z+ a06 x^2 y + a07 x^2 z^2+a08 x^2 z+a09 x^2+
a10 x y^3+a11 x y^2 z+ a12 x y^2+a13 x y z^2+a14 x y z+
a15 x y + a16 x z^3 + a17 x z^2 + a18 x z + a19 x+
a20 y^4 + a21 y^3 z + a22 y^3+ a23 y^2 z^2 +a24 y^2 z+
a25 y^2 + a26 y z^3 + a27 y z^2 + a28 y z + a29 y+
a30 z^4 + a31 z^3 + a32 z^2 + a33 z + a34 = 0
```

Per dichiarare una superficie di 4° ordine, è necessario che ciascuno dei coefficienti (a0...a34) sia in un vettore di 35 termini.

Per esempio, definiamo un toro nel modo complicato. Un toro può essere rappresentato dall'equazione :

```
x^4 + y^4 + z^4 + 2 x^2 y^2 + 2 x^2 z^2 + 2 y^2 z^2 -
2 (r_0^2 + r_1^2) x^2 + 2 (r_0^2 - r_1^2) y^2 -
2 (r_0^2 + r_1^2) z^2 + (r_0^2 - r_1^2)^2 = 0
```

Dove r_0 è il raggio maggiore del toro, la distanza tra il centro del foro della 'ciambella' e la metà dell'anello e r_1 è il raggio minore del toro, la distanza cioè tra la metà dell'anello e la superficie esterna. L'equazione definisce un toro che ha raggio maggiore 6.3 e raggio minore 3.5 (il diametro è così appena inferiore a 20).

```
// Toro di raggio maggiore sqrt(40) e raggio minore sqrt(12)

quartic {
< 1, 0, 0, 0, 2, 0, 0, 2, 0,
-104, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 1, 0, 0, 2, 0, 56, 0,
0, 0, 0, 1, 0, -104, 0, 784 >
sturm
bounded_by { // bounded_by accelera il rendering
sphere { <0, 0, 0>, 10 }
}
}
```

Polinomiali, cubiche e quartiche sono come e quadriche, nel senso che non è necessario capirle per usarle. Il file **shapesq.inc** ha molte quartiche predefinite. La sintassi necessaria per usare una quartica predefinita è :

```
object { Nome_Quartica}
```

Le polinomiali usano calcoli molto complessi e non sempre risulteranno perfette. Se la superficie non è regolare, ha punti disposti in modo strano o casuale, prova ad usare la parola chiave `sturm` nella definizione. Ciò farà eseguire a POV-Ray dei calcoli più lenti, ma più accurati. Normalmente, ma non sempre, ciò risolverà il problema. Se `sturm` non funziona, prova a ruotare o traslare l'oggetto di poco. Vedi la sottodirectory **math** nella directory dei file scena per esempi dell'uso delle polinomiali nelle scene.
Ci sono talmente tante quartiche così diverse, che non possiamo nemmeno iniziare ad elencarle tutte. Se l'argomento ti interessa e sei portato per la matematica, un ottimo libro di riferimento per curve e superfici, dove troverai altre formule di quartiche, è il seguente :

```
"The CRC Handbook of Mathematical Curves and Surfaces"
David von Seggern
CRC Press, 1990
```

7.5.4.3 Quadrica

le superficie quadriche possono creare oggetti come ellissoidi, sfere, coni, cilindri, paraboloidi (per esempio specchi parabolici) e iperboloidi (forme a sella o clessidre). Non confondete le *quartiche* con *quadriche*. Una quadrica è una polinomiale di secondo grado mentre una quartica è di quarto grado. Le quadriche assicurano una maggiore velocità di rendering e incorrono in meno errori.
Una quadrica è definita così :

```
quadric { <A,B,C>, <D,E,F>, <G,H,I>, J }
```

dove le lettere da A a J rappresentano i valori che definiscono i punti x, y, e z che formano la superficie. Questi punti rispondono all'equazione

```
A x^2 + B y^2 + C z^2 +
D xy + E xz + F yz +
G x + H y + I z + J = 0
```

Valori diversi di A, B, C... daranno oggetti diversi. Se prendi un qualunque punto x, y e z e usi le sue coordinate in questa equazione, il risultato sarà uguale a zero se il punto appartiene alla superficie dell'oggetto, negativo se si trova dentro all'oggetto e positivi se si trova fuori. Ecco alcuni esempi :

```
X^2 + Y^2 + Z^2 - 1 = 0 Sfera
X^2 + Y^2 - 1 = 0 Cilindro infinito lungo l'asse z
X^2 + Y^2 - Z^2 = 0 cono infinito lungo l'asse z
```

Il modo più semplice di usare questi oggetti è quello di includere il file **shapes.inc** nella tua scena. Questo file contiene alcune quadriche predefinite, che è possibile posizionare, ruotare, ridimensionare all'interno della scena. È possibile richiamarle usando la seguente sintassi :

```
object { Nome_della_Quadrica }
```

Queste quadriche predefinite sono centrate nell'origine ed hanno raggio unitario. Non confondete il raggio con la larghezza dell'oggetto. Il raggio è la metà del diametro (o larghezza), quindi le normali quadriche sono ampie due unità.
Alcune delle quadriche predefinite sono :

```
Ellipsoid
Cylinder_X, Cylinder_Y, Cylinder_Z
QCone_X, QCone_Y, QCone_Z
Paraboloid_X, Paraboloid_Y, Paraboloid_Z
```

Per una lista completa vedere il file **shapes.inc**.

7.5.5 Geometria Solida Costruttiva

POV-Ray include le operazioni di Geometria solida Costruttiva. Queste operazioni sono : differenza, intersezione, unione, fusione e inverso. Mentre le prime quattro operazioni sono binarie, in altre parole hanno bisogno di due argomenti, l'inverso è un'operazione unaria, necessita di un solo argomento.

7.5.5.1 La CSG

La geometria solida costruttiva è una tecnica per combinare due o più oggetti per creare un nuovo oggetto usando i tre operatori booleani : unione, intersezione e negazione. Queste operazioni funzionano solo su oggetti solidi, cioè oggetti che hanno la parte interna definita. Questo è il caso di tutti gli oggetti descritti nei paragrafi "Primitive Solide Finite" e "Primitive Solide Infinite".
Gli oggetti CSG possono essere utilizzati ovunque si sia fatto uso di un oggetto standard, anche

all'interno dell'oggetto CSG stesso. Possono essere spostati, ruotati e ridimensionati come un qualunque oggetto sia gli oggetti componenti che l'intero oggetto.

7.5.5.2 Dentro e Fuori

Molte primitive, come sfere, parallelepipedi e blob dividono il 'mondo' in due regioni : una regione è *all'interno* dell'oggetto, l'altra all'esterno. Dato un qualunque punto nello spazio, si può stabilire facilmente se è all'interno o all'esterno di una primitiva. Potrebbe essere esattamente sulla superficie, ma in questo caso sarebbe piuttosto problematico determinare dov'è per ragioni numeriche. Anche i piani hanno un 'dentro' e un 'fuori'. Per definizione, la normale alla superficie di un piano è rivolta all'esterno. E' da notare che i triangoli e gli oggetti basati su essi, come 'mesh' e superfici di Bezier, non possono essere usati all'interno di oggetti CSG perché non hanno un interno ed un esterno definiti.
La geometria CSG utilizza i concetti di interno ed esterno per combinare oggetti come spiegato nei paragrafi seguenti.
Immaginate di avere oggetti che si sovrappongono parzialmente, come è mostrato nella figura seguente. Si possono distinguere quattro tipi diversi di punti : i punti che non sono né in A né in B, quelli che sono in A ma non in B, quelli che sono in B ma non in A, e quelli che sono sia in A che in B.

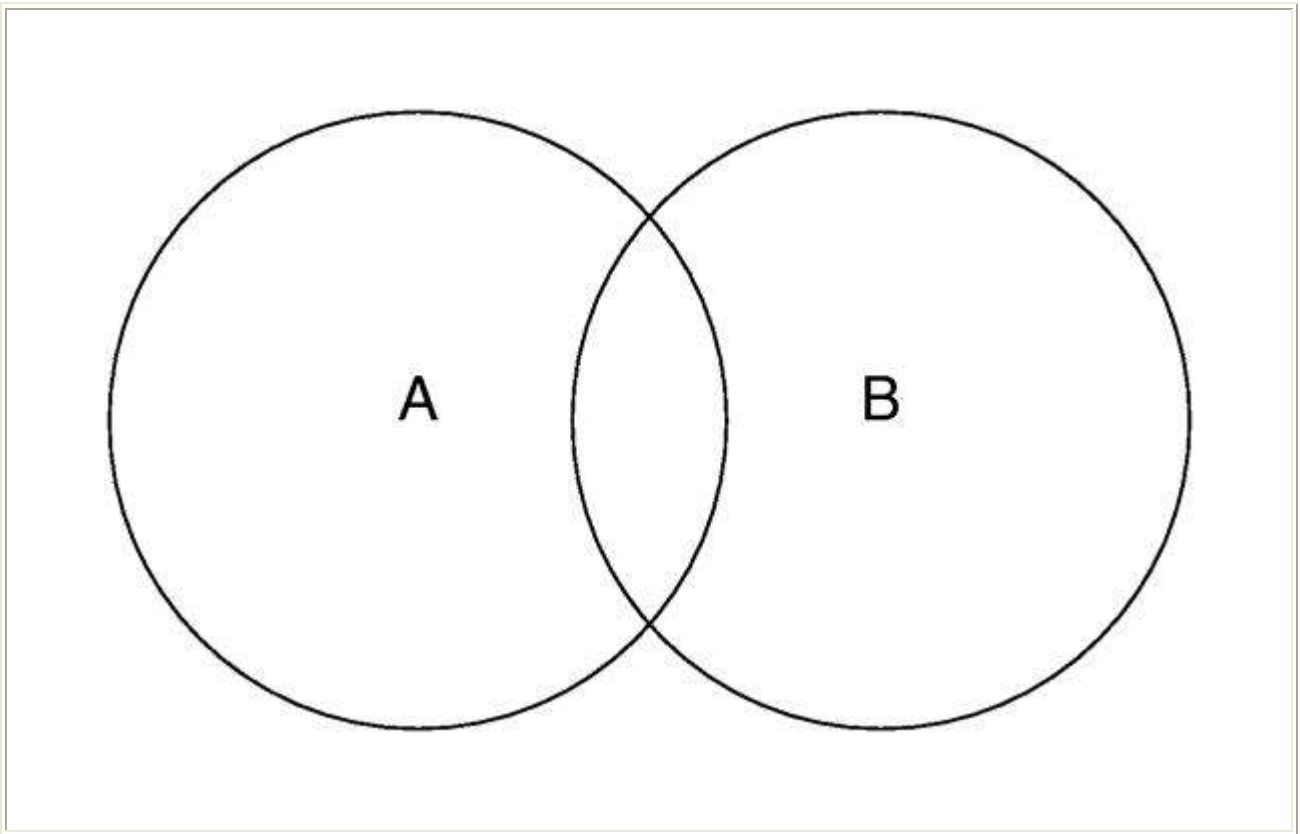



Fig. 206-Due oggetti chi si sovrappongono

Ricordarsi di questo faciliterà la comprensione del funzionamento degli oggetti CSG.

7.5.5.3 Inverso

Quando si usano gli oggetti CSG è spesso utile *invertire* un oggetto in modo che il suo interno si trovi all'esterno. Ciò cambia il modo in cui POV-Ray lo calcola, ma non il suo aspetto. Quando si usa la parola chiave `inverse` l'interno dell'oggetto diventa il suo esterno e viceversa.

Nota che l'operazione di differenza è eseguita intersecando il primo oggetto con la 'negazione' del secondo.

7.5.5.4 Unione

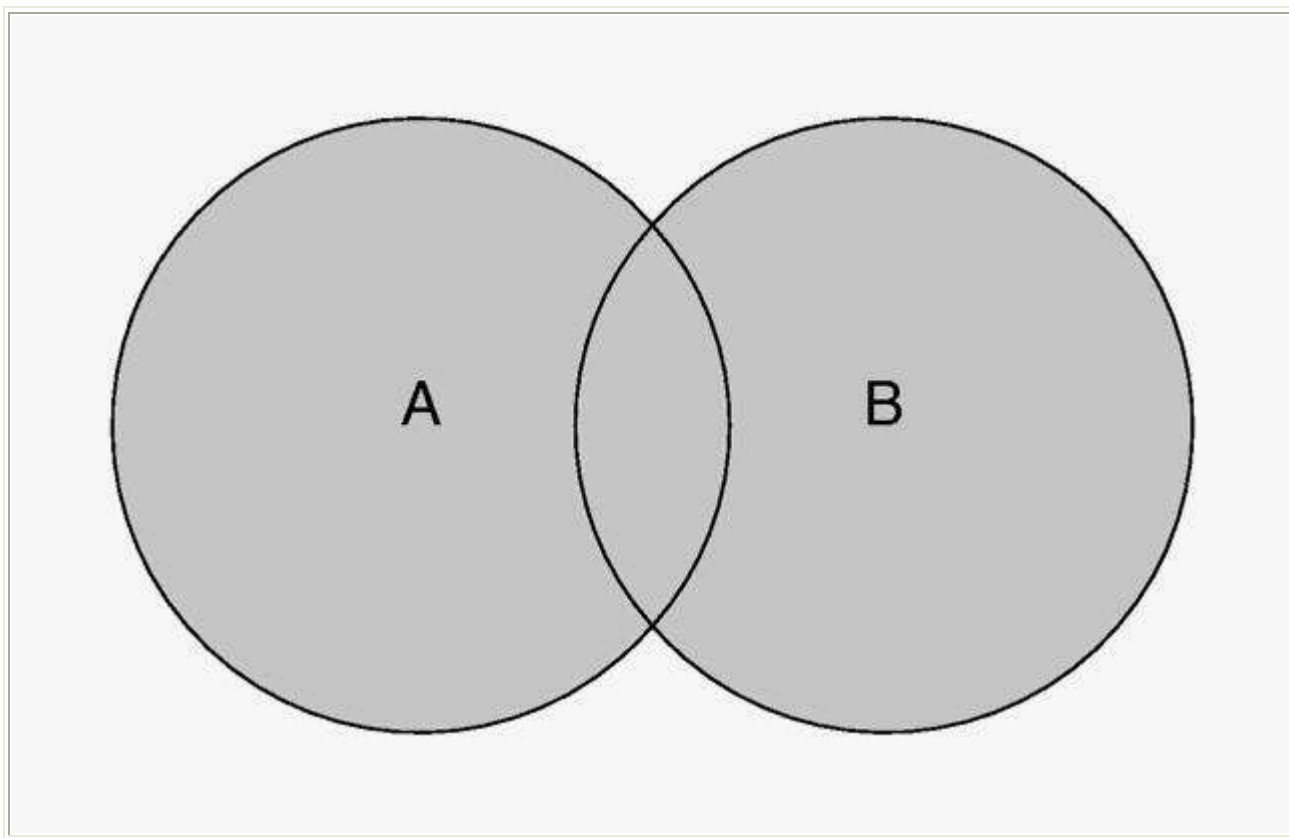



Fig. 207-L'unione di due oggetti

Le frasi `union{...}` sono semplicemente 'colla' utilizzata per legare insieme due o più oggetti in un nuovo oggetto risultante, detto *unione*, che può essere manipolato come una singola entità. L'immagine sopra mostra l'unione degli oggetti A e B. Il nuovo oggetto creato con l'operazione di unione può essere scalato, spostato o ruotato. L'intera unione può avere un'unica texture, ma è comunque possibile assegnare ad ogni oggetto che la compone una texture individuale, che sovrascriverà le eventuali frasi di texture assegnate all'unione.

Le superfici all'interno dell'unione non saranno rimosse e, come si può capire dalla figura, ciò può rappresentare un problema per gli oggetti trasparenti. Se desideri che le superfici condivise tra gli oggetti che fanno parte dell'unione siano rimosse, è necessario usare l'operazione `merge{...}`, che sarà spiegata nel paragrafo "Fusione". La seguente unione è composta da un parallelepipedo ed una sfera :

```
union {
box { <-1.5, -1, -1>, <0.5, 1, 1> }
cylinder { <0.5, 0, -1>, <0.5, 0, 1>, 1 }
}
```

Le precedenti versioni di POV-Ray avevano alcune restrizioni riguardo alle unioni, e spesso era necessario combinare gli oggetti usando la frase `composite{...}`. Questa sintassi non è più necessaria anche se è stata mantenuta la compatibilità all'indietro, ma vi raccomandiamo di usare la frase `union{...}` dal momento che per le versioni future non sarà più garantita la compatibilità con `composite`.

7.5.5.5 Intersezione

Un punto si trova all'interno di un'intersezione se è all'interno di entrambi gli oggetti A e B come mostra la figura qui sotto :

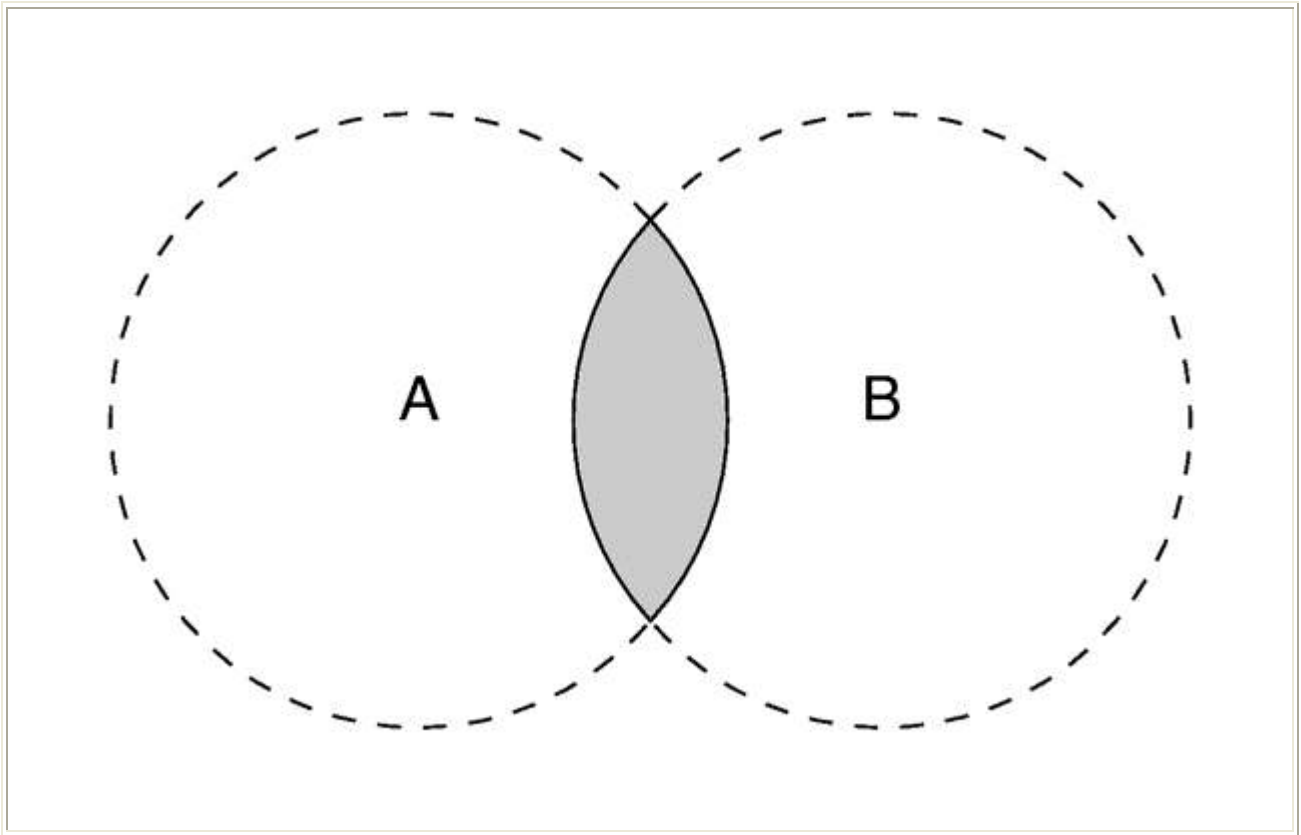



Fig. 208-L'intersezione di due oggetti

per esempio :

```
intersection {
box { <-1.5, -1, -1>, <0.5, 1, 1> }
cylinder { <0.5, 0, -1>, <0.5, 0, 1>, 1 }
}
```

7.5.5.6 Differenza

L'operazione CSG di differenza (parola chiave `difference`) calcola l'intersezione tra il primo oggetto e la negazione del secondo oggetto. Quindi, solo i punti all'interno dell'oggetto A ed esterni all'oggetto B le appartengono.
Il risultato è una sottrazione del secondo oggetto dal primo come mostrato nella figura che segue.

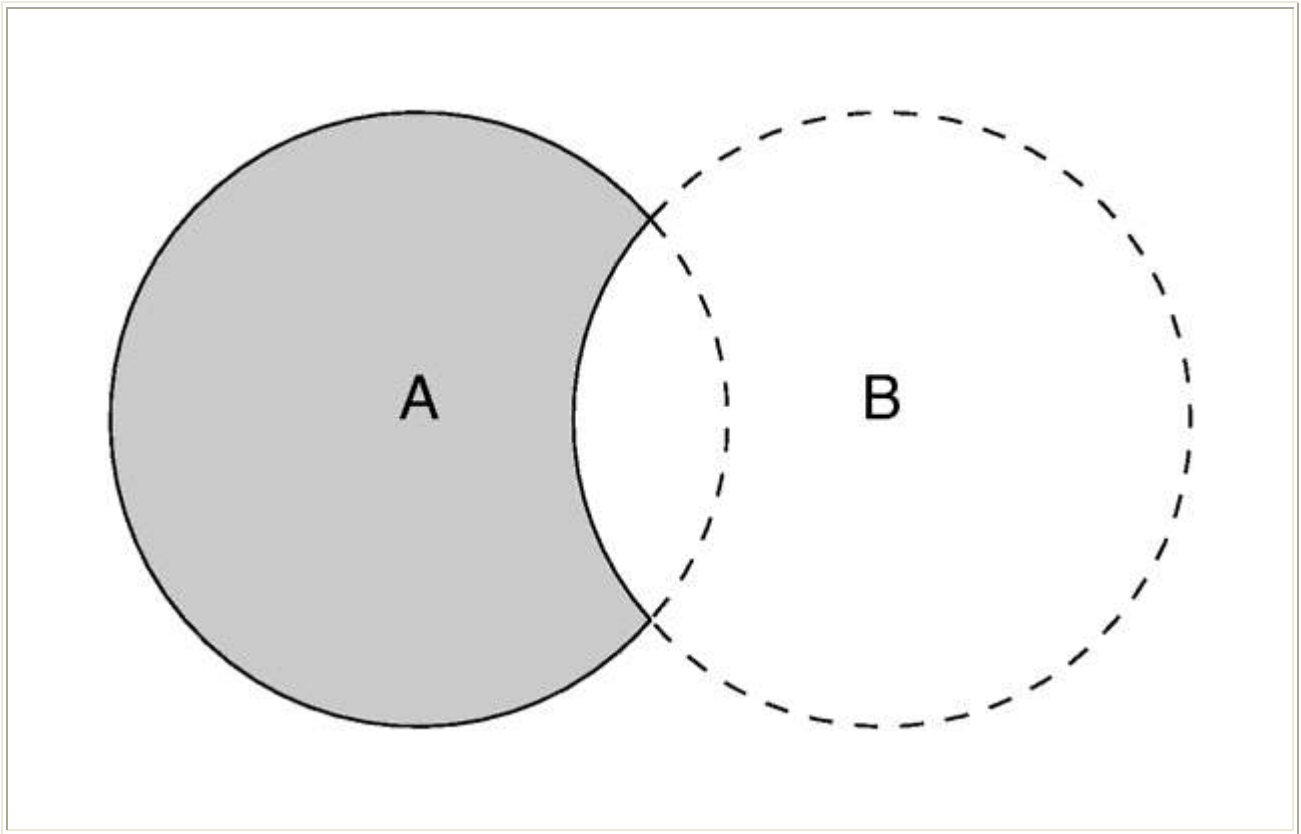



Fig.209-La differenza tra due oggetti

Ad esempio,

```
difference {
box { <-1.5, -1, -1>, <0.5, 1, 1> }
cylinder { <0.5, 0, -1>, <0.5, 0, 1>, 1 }
}
```

7.5.5.7 Fusione (Merge)

L'operazione `union` unisce semplicemente gli oggetti l'uno all'altro, e non rimuove le superfici all'interno dell'unione. Se si usa un'unione trasparente, quelle superfici saranno visibili. L'operazione `merge` evita questo problema. Funziona come l'unione ma elimina le superfici interne come è mostrato dalla figura.

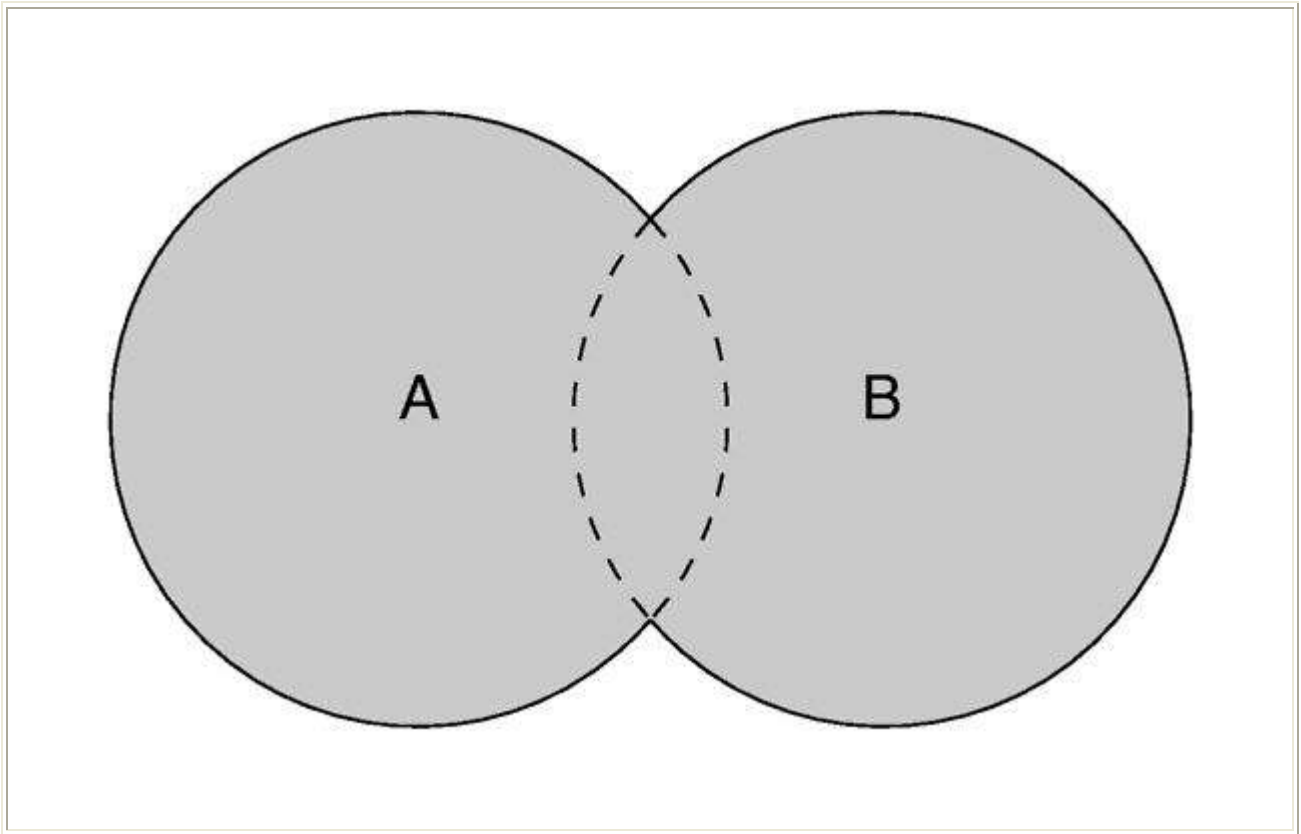



Fig.210-La fusione di due oggetti

7.5.6 Sorgenti Luminose

L'ultimo tipo di oggetto trattato sono le fonti di luce (`light_source`). Le fonti di luce non hanno un contorno visibile. Sono solamente punti che emettono luce. La sintassi completa relativa alle sorgenti luminose è :

```
light_source {
<posizione>
color <colore>
[ spotlight ]
[ point_at <puntata verso> ]
[ radius raggio ]
[ falloff FALLOFF ]
[ tightness apertura ]
[ area_light <asse 1>, <asse2>, dimensione1, dimensione 2 ]
[ adaptive adattiva ]
[ jitter JITTER ]
[ looks_like { oggetto } ]
[ fade_distance distanza di decadimento ]
[ fade_power potenza di decadimento ]
[ atmospheric_attenuation (on/off) ]
}
```

I diversi tipi di sorgenti luminose e i relativi modificatori opzionali saranno descritti nelle sezioni seguenti.

7.5.6.1 Luci Puntiformi

Una sorgente luminosa manda luce di un colore specificato in tutte le direzioni. La posizione è specificata dalla parola chiave `location` e il colore dalla parola chiave `color`. La sintassi completa è :

```
light_source {
<posizione>
color <colore>
[ looks_like { oggetto } ]
[ fade_distance distanza di decadimento ]
[ fade_power potenza di decadimento ]
[ atmospheric_attenuation (on/off) ]
}
```

Le altre parole chiave verranno spiegate dopo.

7.5.6.2 Spot

Uno spot è una sorgente luminosa puntiforme, i cui raggi sono contenuti all'interno di un cono. La luce è forte al centro del cono e si attenua ai lati. La sintassi è :

```
light_source {
<posizione>
color <colore>
spotlight // identifica il tipo di sorgente
point_at <puntata verso>
radius raggio
falloff FALLOFF
tightness apertura
[ looks_like { oggetto } ]
[ fade_distance distanza di decadimento ]
[ fade_power potenza di decadimento]
[ atmospheric_attenuation (on/off) ]
}
```

Lo spot è identificato dalla parola chiave `spotlight`, è posizionato a `location` ed è rivolto verso `point_at`. La seguente illustrazione può essere di aiuto per capire meglio le relazioni tra questi valori.

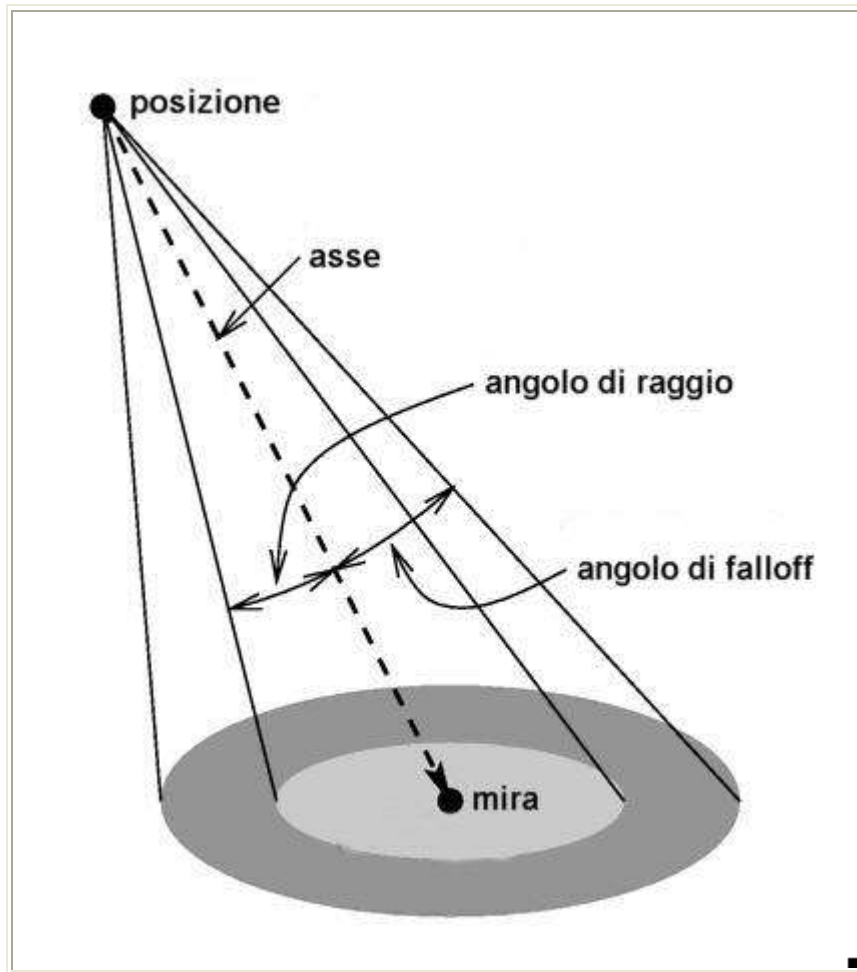



Fig. 211-Luce Spot

Altri parametri sono `radius`, `falloff` e `tightness`.
Per visualizzare mentalmente come sia fatto uno spot basta pensare a due coni uno dentro l'altro. Il cono all'interno è specificato dalla parola chiave `radius` ed è completamente illuminato. Il cono all'esterno è il cono di falloff al di là del quale non c'è luce. I valori per questi due parametri sono la metà dell'angolo di apertura dei coni corrispondenti, entrambi gli angoli dovranno essere minori di 90°. La luce decade lentamente a partire dal raggio del cono interno fino al raggio del cono esterno, come mostrato nella figura (il raggio non deve essere negativo).

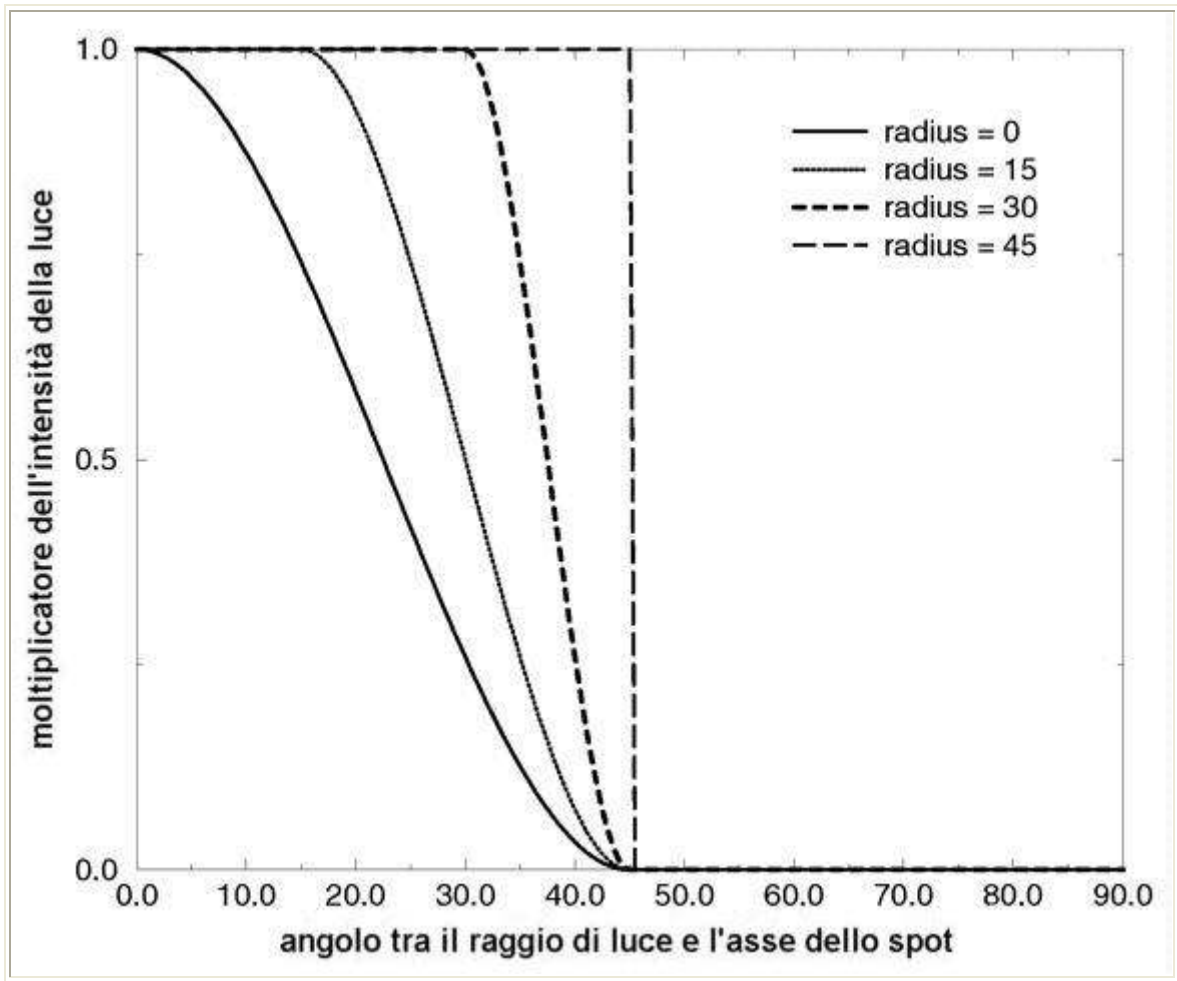



Fig. 212-Curva di intensità relativa della luce con angolo di falloff fissato a 45°

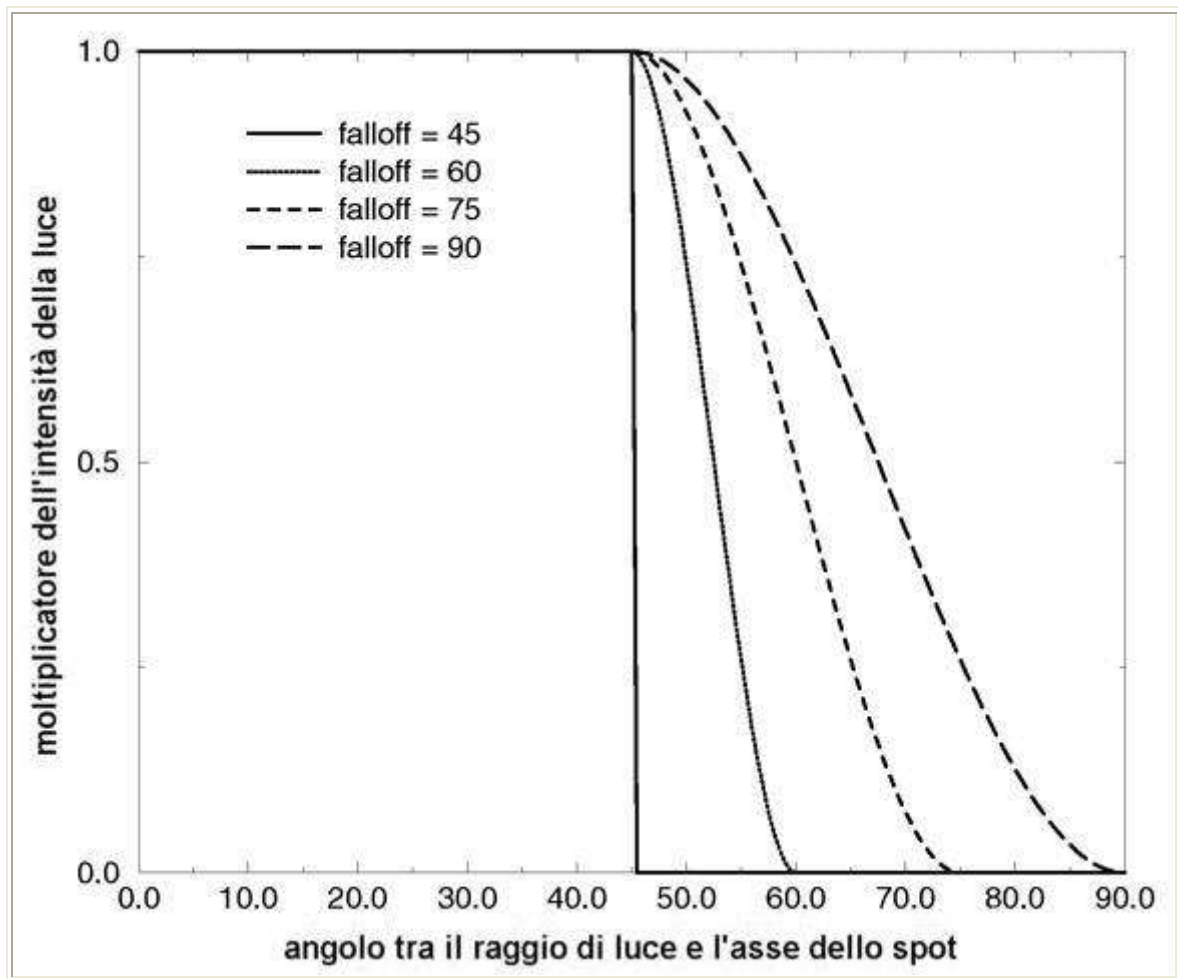



Fig. 213-Curva di intensità relativa della luce con angolo di raggio fissato a 45°

La parola chiave `tightness` specifica la velocità di falloff della luce, dal centro verso la fine del cono di falloff (all'esterno sarà buio). Il valore predefinito per questa parola chiave sarà di 10. Valori minori renderanno più luminosa la luce allargando il cono e renderanno i contorni più netti. Valori più bassi renderanno la luce più tenue, lo spot più piccolo e i contorni meno marcati. Valori da 1 a 100 sono accettabili.

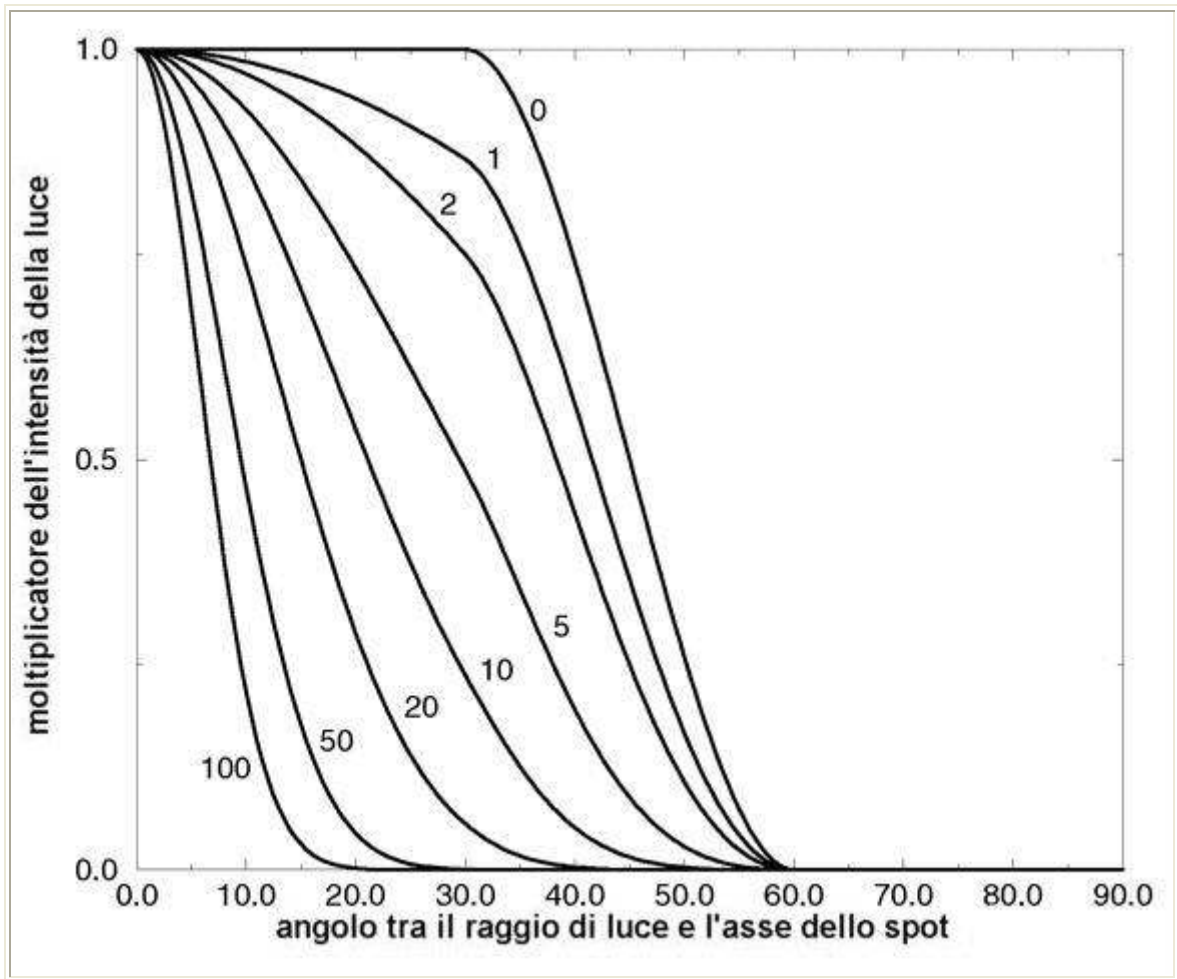



Fig. 214-Curva di intensità relativa della luce con raggio fissato a 30° e falloff a 60° e diversi valori di tightness

Dalla figura si può notare come il raggio e il falloff interagiscono con l'apertura. Solo valori negativi del raggio daranno al valore di `tightness` pieno controllo sull'aspetto della luce, come si può vedere dalla figura sottostante. In questo caso il valore dell'angolo di falloff non avrà alcun effetto e l'area illuminata sarà determinata solo dal parametro `tightness`.

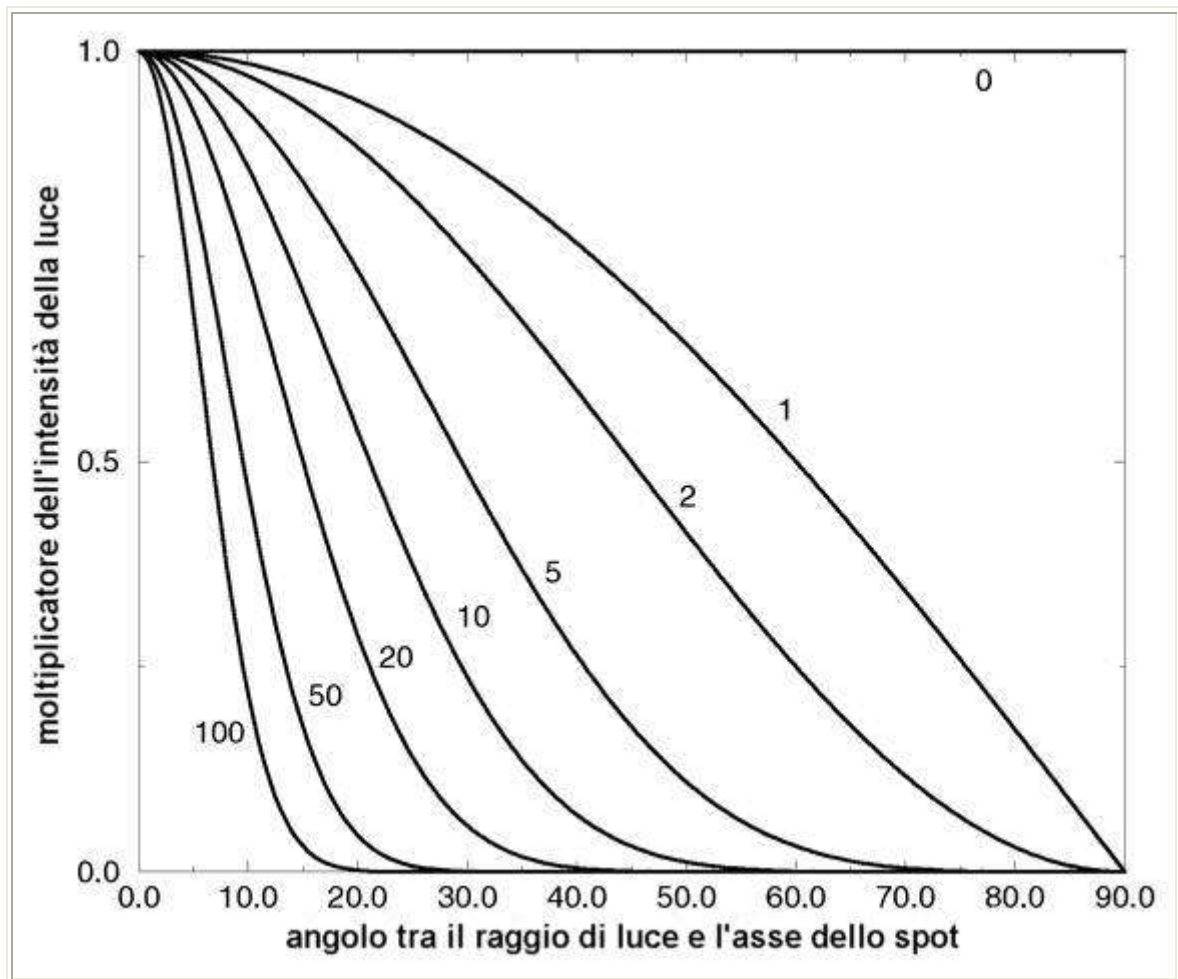



Fig. 215-Curva di intensità relativa con raggio negativo e tightness variabile

Gli spot possono essere utilizzati in ogni situazione in cui si usi una normale fonte di luce. Come tutte le sorgenti luminose, essi sono invisibili. Sono considerati come oggetti e possono essere inclusi in oggetti CSG. Possono anche essere utilizzati insieme alle area light.

7.5.6.3 Luci Cilindriche

Le luci cilindriche funzionano in maniera molto simile agli spot, ma i raggi di luce sono contenuti in un cilindro e non in un cono. La sintassi è :

```
light_source {
<posizione>
color <colore>
cylinder
point_at <puntata verso>
radius raggio
falloff FALLOFF
tightness apertura
[ looks_like { oggetto} ]
[ fade_distance distanza di decadimento ]
[ fade_power potenza di decadimento]
[ atmospheric_attenuation (on/off)]
}
```

Le parole chiave `radius`, `falloff` e `tightness` si comportano come per gli spot.

Si dovrebbe ricordare che le luci cilindriche sono comunque delle sorgenti luminose puntiformi. I raggi sono emessi da un punto e contenuti in un cilindro, ma non sono paralleli (e quindi non si comportano come un laser, N.d.T.).

7.5.6.4 Area Light

Le *area light* (sorgenti di luce estese) occupano un'area finita di spazio e hanno una o due dimensioni. Sono in grado di creare ombre sfumate.

Le area light usate da POV-Ray sono di forma rettangolare, come un pannello di luci. Piuttosto che usare un modello di calcolo piuttosto complesso per modellare un'area light effettiva, questa è stata approssimata ad una sorgente puntiforme che si diparte dall'area occupata dalla luce stessa. L'intensità di ogni singolo punto di luce che è compreso nell'area light è quindi ridotta, cosicché il valore totale di luce emessa risulterà uguale al colore della luce specificato. La sintassi è :

```
light_source {
<posizione>
color <colore>
area_light <asse1>, <asse2>, dimensione1, dimensione2
adaptive adattativa
jitter jitter
[ spotlight ]
[ point_at <punta verso> ]
[ radius raggio ]
[ falloff FALLOFF ]
[ tightness apertura ]
[ looks_like { oggetto } ]
[ fade_distance distanza di decadimento ]
[ fade_power potenza di decadimento ]
[ atmosphere (on/off) ]
[ atmospheric_attenuation (on/off) ]
}
```

La posizione e il colore della luce si specificano come per gli altri tipi di luce.

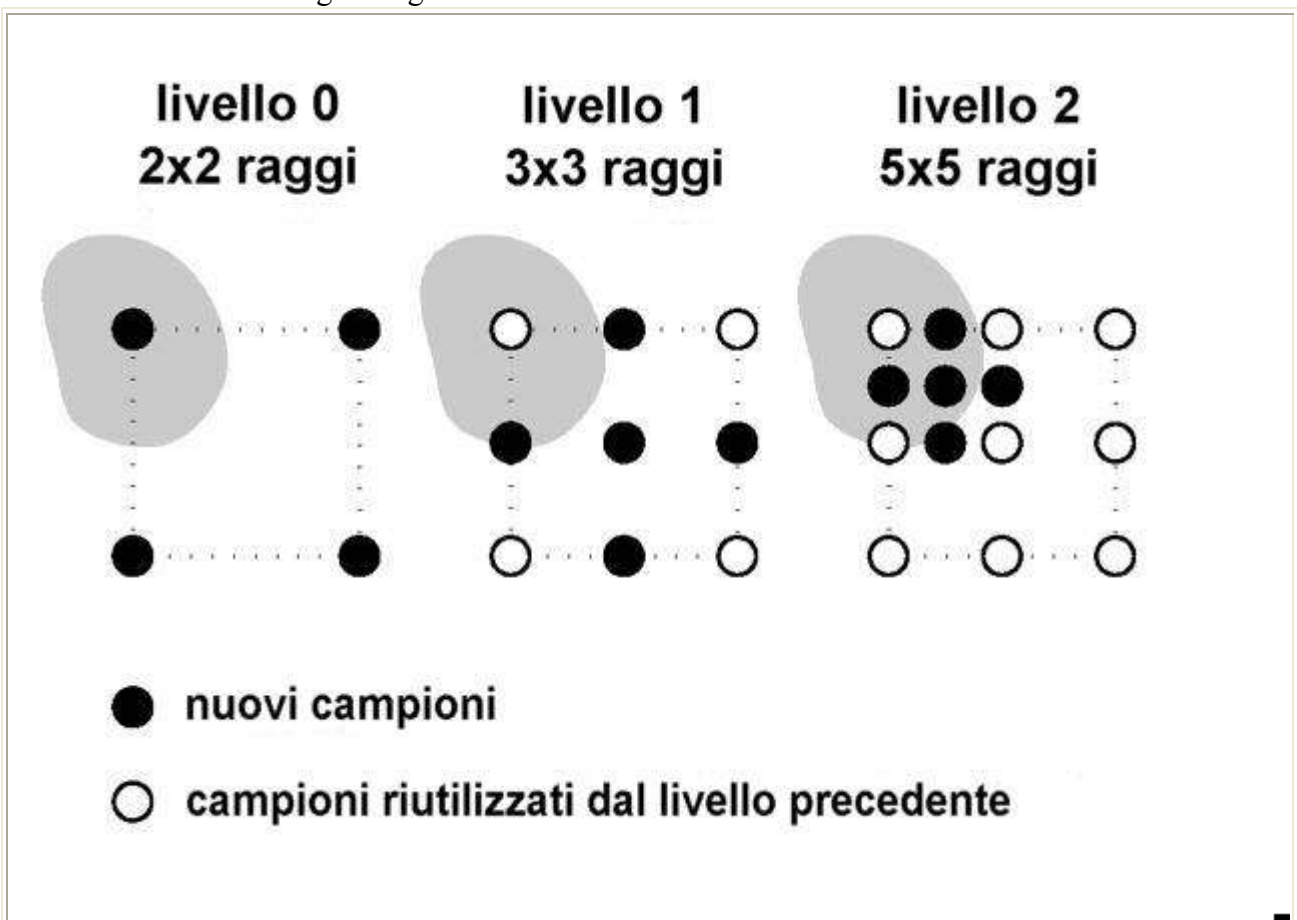
Il comando `area_light` definisce la dimensione e l'orientamento della luce così come il numero delle luci. I vettori `asse1` e `asse2` specificano la lunghezza e la direzione del perimetro che contiene le luci. Poiché le area light hanno forma rettangolare, questi vettori dovranno essere fra loro perpendicolari. Più grande sarà la dimensione della luce più morbide saranno le ombre. I numeri `dimensione1` e `dimensione2` specificano il numero di luci per lato del rettangolo, aumentando il quale si sfumano le ombre, ma si allunga considerevolmente il tempo necessario per il rendering.

Il comando `jitter` è facoltativo. Quando è usato causa un piccolo spostamento nella posizione delle luci all'interno del rettangolo definito dall'area light, in modo da evitare zone d'ombra. Questo parametro è completamente casuale e quindi è meglio non utilizzarlo nelle animazioni.

E' possibile anche definire parametri appartenenti agli spot e includerli nei parametri delle area light, per creare *area spotlights*, ovvero dei gruppi di spot estesi su un'area. Questo tipo di luce è utilizzato in genere per aumentare la velocità di rendering delle scene che usano le area light, dal momento che è possibile limitare le aree di calcolo delle ombre solo a quelle parti realmente incluse nell'inquadratura.

Un effetto interessante può essere creato usando sorgenti di luce lineari. Piuttosto di definire spazi rettangolari, si possono distribuire le luci lungo una linea creando una sorta di tubo luminoso. Per fare questo basta che una delle dimensioni specificate sia 1.

Il comando `adaptive` è usato per attivare il sovracampionamento adattivo della luce. Di default POV-Ray calcola la quantità di luce che colpisce una superficie tracciando un raggio a partire da ogni punto all'interno del rettangolo definito. Questo metodo come è possibile capire è molto lento. Il metodo adattivo tende ad approssimare questo processo usando un numero minore di raggi. Il numero specificato dopo la parola chiave indica in che proporzione si debba usare il metodo adattivo. Più alto sarà quel valore, più accurate saranno le ombre e più lento il rendering. Se non si è sicuri nella scelta di un valore ragionevole, un buon punto di partenza potrebbe essere `adaptive 1`. La parola chiave `adaptive` accetta solo valori interi e non può essere impostata con numeri minori di zero.

Quando esegue il campionamento adattivo, POV-Ray inizia lanciando un raggio di prova ai quattro angoli dell'area light. Se la quantità di luce ricevuta da tutti e quattro gli angoli è approssimativamente la stessa, allora l'area light può essere interamente visibile, o interamente coperta da qualche oggetto. L'intensità della luce è quindi calcolata come la media delle intensità della luce ricevuta dai quattro angoli. Se invece l'intensità della luce che giunge ai quattro angoli è significativamente diversa, allora POV-Ray interpreta che l'area light è *parzialmente* coperta. L'area light è quindi divisa in altri quattro quarti ed ogni sezione è campionata come descritto sopra. Ciò permette a POV-Ray di calcolare rapidamente quanta parte dell'area light è inclusa nell'inquadratura senza avere bisogno di tracciare un raggio per ogni luce dell'insieme. Il campionamento funziona come è mostrato nella figura seguente

.Fig. 216- Sovracampionamento adattivo nelle area light

Anche se il metodo di campionamento adattivo è veloce (relativamente parlando), può comunque creare ombre non molto accurate. La soluzione è di ridurre la quantità di campionamento adattivo senza disattivarlo completamente. Il numero dopo la parola chiave `adaptive` imposta il numero di sezioni in cui l'area light sarà divisa prima che inizi la fase di campionamento adattivo vera e propria. Per esempio, se usi `adaptive 0` saranno tracciati un minimo di quattro raggi. Se usi `adaptive 1` saranno tracciati un minimo di nove raggi (`adaptive 2` darà 25 raggi, `adaptive 3` 81 e così via). Ovviamente, più raggi sono tracciati, più il rendering sarà lento. E' quindi consigliabile usare il valore più basso che dia risultati accettabili.

Il numero di raggi non supererà mai il valore che viene specificato per i punti che definiscono le righe e le colonne del rettangolo in cui è contenuta l'area light. Per esempio, `area_light x,y,4,4` specifica un insieme 4 per 4 di luci. Se specifichi `adaptive 3` allora dovrai iniziare con un insieme di 9 per 9 luci, come minimo. In questo caso non sarà eseguito il campionamento adattivo e verrà usato un insieme 4 per 4.

7.5.6.5 Luci Senza Ombra

Usando la parola chiave `shadowless` si impedisce alla sorgente luminosa di proiettare ombre.

7.5.6.6 Looks_like

Normalmente la sorgente luminosa è invisibile. La luce viene semplicemente irradiata da un punto o da un'area invisibili. Puoi assegnare ad una sorgente luminosa una qualunque forma, aggiungendo una frase

```
looks_like{oggetto}.
```

Usando `looks_like`, implicitamente viene applicata all'oggetto anche l'istruzione `no_shadow` per impedire che la luce sia coperta dall'oggetto stesso. Senza questo comando, la luce dentro all'oggetto non potrebbe illuminare la scena e l'oggetto proietterebbe in effetti la propria ombra su tutto. Se vuoi che l'oggetto al quale viene associata la luce la intercetti, allora dovrebbe essere unito alla luce da una frase `union{...}` come segue :

```
union {
light_source { <100, 200, -300> color White }
object { Lampadina }
}
```

Presumibilmente, parti della lampadina sono trasparenti per permettere alla luce di uscirne.

7.5.6.7 Attenuazione

Per default, POV-Ray non attenua la luce di nessuna sorgente luminosa in relazione alla distanza. Per ottenere effetti più realistici, si possono utilizzare i comandi `fade_distance` e `fade_power` per simulare il decadimento dell'intensità della luce in relazione alla distanza.
La parola chiave `fade_distance` è usata per specificare la distanza alla quale arriva la piena intensità luminosa, cioè l'intensità che è stata definita con la parola chiave `color`. Il decadimento viene specificato dalla parola chiave `fade_power` che ne determina la velocità. Per esempio, decadimenti lineari o quadratici possono essere usati impostando `fade_power` a 1 o 2

rispettivamente. La formula completa per calcolare il fattore di decadimento è :

$$attenuazione = \frac{2}{1+(d/distanza_di_decadimento)^{potenza_di_decadimento}}$$

Nella quale `d` rappresenta la distanza dalla sorgente luminosa.

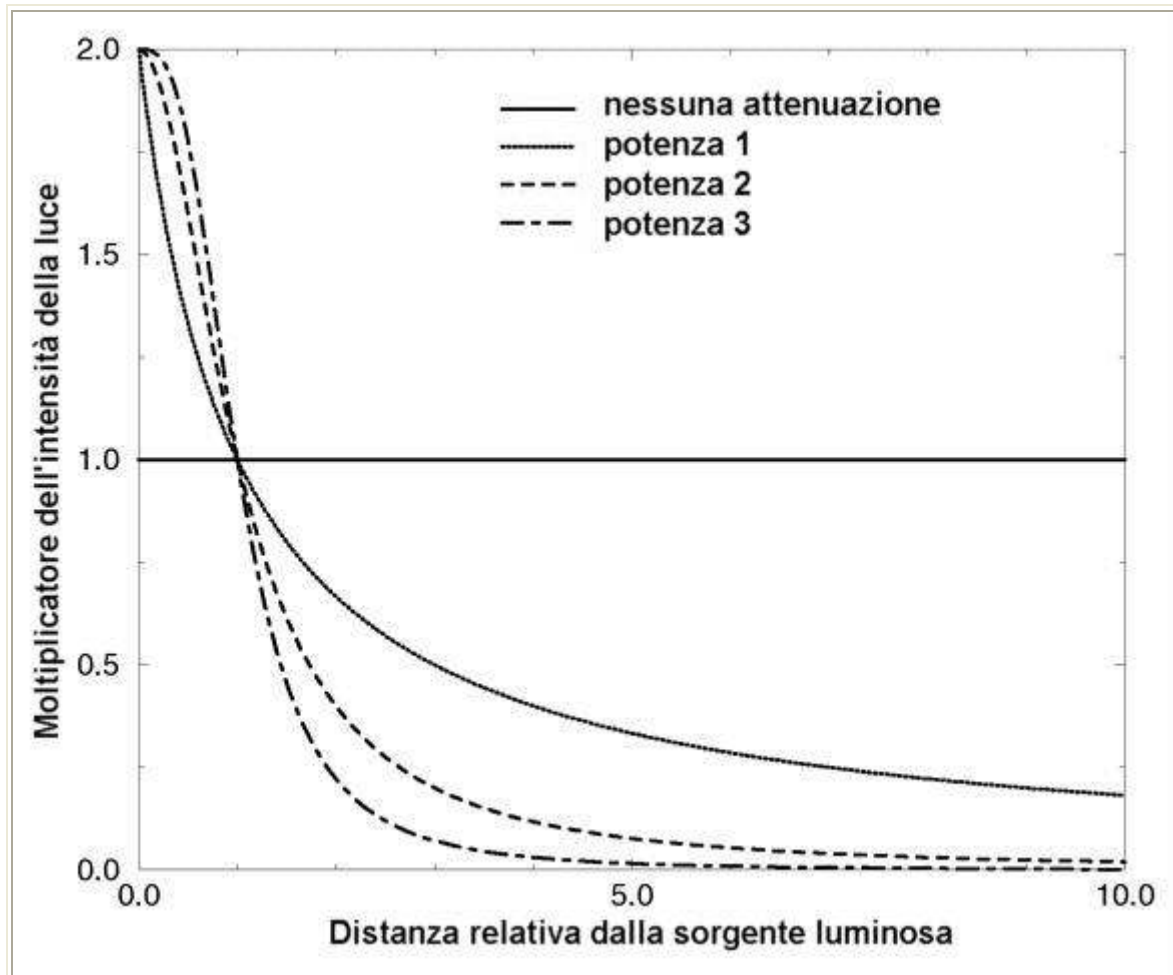



Figura 217- Curva di intensità relativa della luce in funzione della distanza, a valori diversi di fade_power

Si dovrebbero notare due importanti fatti : primo, per valori di `fade_distance` maggiori di 1 l'intensità della luce a distanze minori di `fade_distance` aumenta. Questo è necessario per ottenere la giusta intensità della luce nel punto distante esattamente `fade_distance` dalla sorgente luminosa. Secondo, solo la luce che proviene direttamente dalla sorgente luminosa è attenuata. La luce riflessa o rifratta non è attenuata dalla distanza.

7.5.6.8 Interazione con l'Atmosfera

Per default, le sorgenti luminose interagiranno con un'eventuale atmosfera aggiunta alla scena. Ciò può essere disattivato usando la parola chiave `atmosphere` all'interno delle impostazioni relative alla sorgente luminosa.

```
light_source {
...
atmosphere off
}
```

7.5.6.9 Attenuazione Atmosferica

Normalmente l'intensità della luce che proviene dalle sorgenti luminose non è modificata dalla nebbia o dall'atmosfera. Ciò può essere cambiato attivando il comando `atmospheric attenuation` per una determinata luce. Tutta la luce che proverrà da questa sorgente luminosa, sarà attenuata via via che viaggia attraverso la nebbia o l'atmosfera. Questo determinerà un decadimento esponenziale dipendente dalla distanza e dalle impostazioni della nebbia o dell'atmosfera utilizzate. Se non c'è nebbia o atmosfera non si verificherà alcun cambiamento.

7.5.7 Modificatori di Oggetti

Una grande quantità di modificatori può essere applicata agli oggetti. Trasformazioni come traslare, ruotare e scalare, sono già state discusse. Le modifiche apportabili alle texture sono spiegate in una sezione a sé stante. Qui vengono trattati altri tre importanti modificatori : `clipped_by`, `bounded_by` e `no_shadow`. Sebbene gli esempi che seguono useranno frasi generiche di descrizione degli oggetti e identificatori, questi modificatori possono essere usati su ogni tipo di oggetto, come sfere, parallelepipedi, ecc.

7.5.7.1 Clipped_By

La frase `clipped_by{...}` è tecnicamente un modificatore di oggetti, ma in realtà ha una funzione simile all'intersezione CSG. Ad esempio :

```
object {
Coso
clipped_by{plane{y,0}}
}
```

Ogni parte dell'oggetto Coso che è all'interno del piano è tenuta, mentre le restanti parti sono tagliate via (clipped off). In un'intersezione CSG il foro sarebbe stato chiuso. Con `clipped_by` rimane aperto. Per esempio, la seguente figura mostra l'oggetto A 'clipped_by' l'oggetto B.

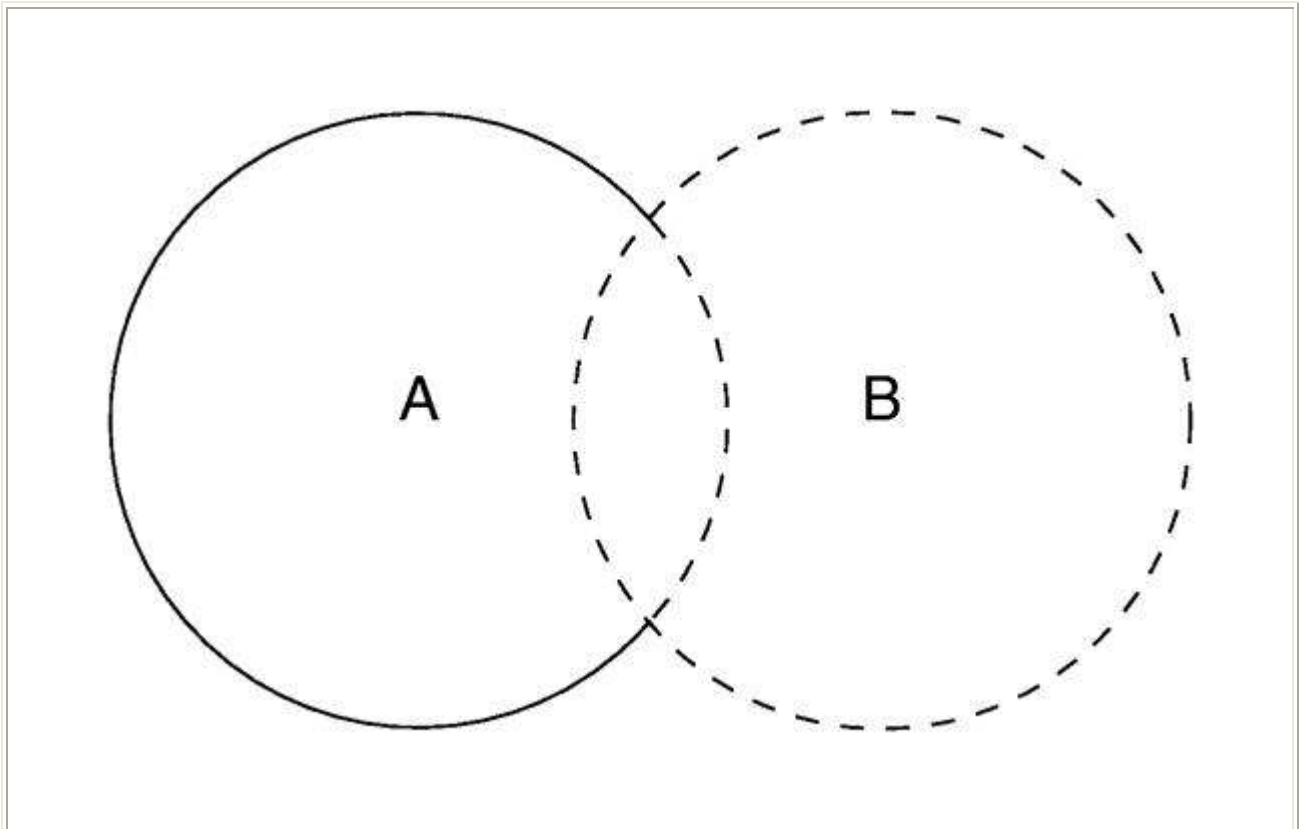



Figura 12- Effetto di clipped_by

Il modificatore `clipped_by` può essere usato per ritagliare porzioni di un oggetto. In molti casi risulterà anche più rapido nel rendering rispetto ad altri metodi di modifica di un oggetto come le operazioni di CSG.
Spesso vorrai usare i modificatori `clipped_by` e `bounded_by` con lo stesso oggetto. Il seguente modo di scrivere risparmierà tempo di battitura ed userà meno memoria.

```
object {
Coso
bounded_by { box { <0,0,0>, <1,1,1> } }
clipped_by { bounded_by }
}
```

Queste righe dicono a POV-Ray di usare lo stesso parallelepipedo per entrambi i comandi `clipped_by` e `bounded_by`.

7.5.7.2 Bounded_By

I calcoli necessari per determinare se un raggio colpisce un oggetto possono impiegare molto tempo. Ogni raggio deve essere provato su ogni oggetto nella scena. POV-Ray cerca di aumentare la velocità del processo costruendo un insieme di scatole invisibili chiamate *bounding boxes* che racchiudono gli oggetti. In questo modo, un raggio che attraversa una parte della scena, non deve essere provato di nuovo su oggetti che si trovano in un'altra parte lontana della scena. Quando un gran numero di oggetti è presente, le scatole sono molto vicine le une alle altre. POV-Ray può usare bounding boxes su ogni oggetto finito ed anche su quadriche. Comunque, agli oggetti infiniti (come piani, quartiche, cubiche e polinomiali) non può essere applicato il bounding *automaticamente*. Il bounding viene applicato automaticamente agli oggetti CSG solo se contengono oggetti finiti (ma in alcuni casi anche se contengono oggetti infiniti). Ciò funziona applicando l'insieme delle operazioni CSG alle bounding boxes per tutti gli oggetti compresi nell'oggetto CSG. Per le operazioni di differenza ed intersezione questo difficilmente porterà ad un bounding ottimale. Qualche volta (in relazione alla complessità dell'oggetto CSG) potrà essere meglio utilizzare la frase `bounded_by{...}` su quegli oggetti.
Normalmente non è necessario usare il comando `bounded_by`, ma ci sono casi in cui usarlo significa anche incrementare la velocità del rendering di oggetti complessi. Gli oggetti che determinano il bounding dicono a POV-Ray che l'oggetto 'bounded' è completamente racchiuso in un oggetto più semplice (che viene definito all'interno della frase `bounded_by{...}`). Durante il rendering, i raggi sono prima provati sull'oggetto delimitante (bounding box). Se i raggi lo colpiscono, il programma eseguirà ulteriori prove sull'oggetto, più complicato, che si trova all'interno dell'oggetto delimitante. Se ciò non avviene, l'intero oggetto complesso sarà evitato con notevole incremento della velocità di rendering. Per usare oggetti delimitanti, basta includere le seguenti linee nella dichiarazione del tuo oggetto :

```
bounded_by {
object { ... }
}
```

Un esempio di oggetto delimitante :

```
intersection {
sphere { <0,0,0>, 2 }
plane { <0,1,0>, 0 }
plane { <1,0,0>, 0 }
```

```
bounded_by { sphere { <0,0,0>, 2 } }
}
```

I migliori oggetti delimitanti sono sfere o parallelepipedi poiché questi oggetti sono molto ottimizzati, comunque ogni oggetto può essere usato. Se lo stesso oggetto delimitante è un oggetto finito che risponde alla suddivisione del bounding, allora anche l'oggetto che racchiude rientrerà all'interno della suddivisione.
Gli oggetti CSG possono trarre molto beneficio dal bounding anche senza il comando `bounded_by`. Comunque si possono presentare problemi in intersezioni, differenze e fusioni (merge). In questi tre tipi di oggetti CSG il bounding automatico coprirà interamente tutti i loro componenti. Comunque, il risultato di queste intersezioni potrà essere un oggetto più piccolo. Confronta le dimensioni per l'unione e l'intersezione nelle illustrazioni della sezione precedente. E' possibile disegnare un parallelepipedo più piccolo intorno all'intersezione di A e B rispetto a quello della loro unione, ma gli oggetti che delimitano A e B saranno comunque grandi come l'unione di A e B a prescindere dal tipo di operazione CSG specificata. E' quasi sempre una buona idea aggiungere manualmente il comando `bounded_by` a intersezioni, differenze e merge ed è spesso meglio **non** applicare il bounding all'unione. Se l'unione non ha componenti del bounding o del clipping POV-Ray può separare i componenti dell'unione ed applicare il bounding automatico a ciascuno dei componenti finiti. Si deve notare che alcune utility come **raw2pov** possono generare il bounding con maggiore efficienza di POV-Ray. Comunque, si può applicare il bounding automatico alla maggior parte delle unioni che possono essere create. Per ragioni tecniche, POV-Ray non può suddividere un oggetto *merge* nei suoi componenti. E' probabilmente meglio applicare il bounding manualmente, soprattutto se l'oggetto è molto complesso. E' da notare che se l'oggetto delimitante è troppo piccolo o in una posizione sbagliata, potrebbe suddividere l'oggetto contenuto in maniere imprevedibili oppure l'oggetto potrebbe non essere visibile. Per eseguire il *clipping* di un oggetto, è necessario usare la parola chiave `clipped_by` come è specificato nel paragrafo precedente. E' possibile che tu voglia usare `clipped_by` e `bounded_by` con lo stesso oggetto. Le frasi che seguono fanno in modo che si possa risparmiare tempo di battitura ed usare meno memoria.

```
object {
Coso
clipped_by{ box { <0,0,0>,<1,1,1 > }}
bounded_by{ clipped_by }
}
```

Questo dice a POV-Ray di usare, come oggetto delimitante, lo stesso parallelepipedo che era stato usato per essere 'ritagliato' da Coso.

7.5.7.3 Hollow (vuoto)

POV-Ray assume che gli oggetti siano costituiti di un materiale che ne riempie completamente l'interno. Aggiungendo la parola chiave `hollow` l'oggetto può diventare cavo. Questa parola chiave è molto utile se si vuole che si verifichino gli effetti determinati dall'atmosfera anche *dentro* all'oggetto. E' anche necessaria per gli oggetti che contengono una halo (vedi "Aloni") per i dettagli. Per rendere cavo un oggetto CSG devi semplicemente aggiungere la parola chiave `hollow` all'oggetto che si trova al livello superiore. Tutti gli oggetti in esso racchiusi diverranno cavi a meno che non sia specificato diversamente. Il seguente esempio imposta entrambe le sfere che si trovano all'interno dell'unione come cave.

```
union {
sphere { -0.5*x, 1 }
sphere { 0.5*x, 1 }
```

```
hollow
}
```

mentre il prossimo esempio considera cava solo la seconda sfera perché per la prima sfera è stato specificato esplicitamente il parametro `hollow off`.

```
union {
sphere { -0.5*x, 1 hollow off }
sphere { 0.5*x, 1 }
hollow
}
```

7.5.7.4 No_Shadow (Senza Ombra)

Si può specificare la parola chiave `no_shadow` nella descrizione di un oggetto per far sì che questo non proietti ombre. Questa funzione è utile per effetti speciali e per creare l'illusione che l'oggetto sia una sorgente luminosa visibile. Questa parola chiave era necessaria nelle precedenti versioni di POV-Ray che non avevano il comando `looks_like`. Adesso, è utile per creare oggetti come raggi laser o altri effetti irreali.
Basta aggiungere la parola chiave come segue :

```
object {
Coso
no_shadow
}
```

7.5.7.5 Sturm

Alcuni degli oggetti di POV-Ray ti permettono di scegliere tra algoritmi veloci, ma talvolta non precisi ed algoritmi più lenti ma più accurati. Questo è il caso di tutti gli oggetti la cui superficie, per essere risolta, richiede la soluzione di un polinomio di terzo o quarto grado. Ci sono soluzioni analitiche per quelle funzioni polinomiali che possono essere usate. Polinomi di grado inferiore sono banali da risolvere, mentre polinomi di gradi superiori al secondo richiedono algoritmi iterativi. Uno di questi algoritmi è il *Polinomio di Sturm*.
La seguente lista mostra tutti gli oggetti per i quali può essere usato il polinomio di Sturm.

```
blob
cubic
lathe (solo con spline quadratiche)
poly
prism (solo con spline cubiche)
quartic
sor
```

7.6 Texture

La texture descrive l'aspetto dell'oggetto, cioè il materiale di cui è fatto. Le texture sono combinazioni di pigmento, normali, finitura ed halo. Il pigmento è il colore o la combinazione di colori del materiale. Le normali sono un metodo per simulare ingrossamenti, increspature, o onde nella superficie modificando il vettore normale ad essa. La finitura descrive le proprietà di riflessione e rifrazione del materiale. L'alone (halo) simula effetti come nuvole, nebbia, fuoco, ecc. usando una distribuzione di particelle in un campo di densità definito all'interno dell'oggetto.

Una texture semplice è formata da un singolo pigmento e volendo, da normali, finitura e da uno o più aloni. Una texture speciale combina due o più texture usando funzioni per creare motivi, o per fonderle insieme. Le texture speciali possono essere rese molto complesse annidando motivi all'interno di motivi. Comunque i livelli più interni sono sempre costituiti da texture semplici. E' da notare che nonostante che noi chiamiamo *semplice* una texture, questa può essere anche molto complessa. Il termine semplice significa soltanto che ha un solo pigmento, normali, finitura, halo.

La forma più completa per definire una texture semplice è :

```
texture {
IDENTIFICATORE_DI_TEXTURE
pigment {...}
normal {...}
finish {...}
halo {...}
TRASFORMAZIONI
}
```

Ognuno dei termini all'interno di una texture è facoltativo, ma se sono presenti degli identificatori questi devono essere inseriti per primi e le trasformazioni devono essere scritte per ultime. I parametri riguardanti i pigmenti, le normali, le finiture modificano qualunque parametro riguardante pigmenti, normali e finiture precedentemente specificato all'interno degli identificatori di texture. Gli aloni sono aggiunti agli aloni già esistenti. Se non è specificato nessun identificatore di texture, i parametri dei pigmenti, delle normali e delle finiture modificano i valori correnti e tutte le halo sono aggiunte alla halo predefinita, se c'è. Le trasformazioni sono traslazioni, rotazioni, ridimensionamento e matrici. Devono essere specificate per ultime.
I paragrafi seguenti descrivono tutte le opzioni disponibili per quanto riguarda tutti i pigmenti, le normali, le finiture e gli aloni. Le texture speciali sono spiegate più avanti.

7.6.1 Pigmento

Il colore o la disposizione di più colori per un oggetto sono definiti da una frase `pigment`. Tutte le texture semplici devono avere un pigmento. Se non ne viene specificato uno, sarà usato quello predefinito. Una frase relativa al pigmento è una parte delle specificazioni di una texture. Comunque può essere noioso usare frasi relative alle texture solo per aggiungere un colore ad un oggetto. Quindi si può aggiungere un pigmento direttamente all'oggetto senza specificare esplicitamente che questo è una parte della texture. Per esempio :

```
//questo...

object {
Coso
texture {
pigment {color Red}
}
}


//si può abbreviare con...

object{
  Coso
```

```
  pigment { color Red }
}
```

Il colore che viene specificato sarà quello che l'oggetto avrà quando sarà completamente illuminato. Si prende il colore base inerente all'oggetto e POV-Ray lo schiarirà o scurirà in relazione all'illuminazione della scena. Il parametro è chiamato `pigment` perché si sta definendo il colore base dell'oggetto piuttosto che il suo vero aspetto.
La forma più completa per definire un pigmento è la seguente :

```
pigment {
IDENTIFICATORE_DEL_PIGMENTO
DISPOSIZIONE_DEI_COLORI
MODIFICATORI_DEL_PIGMENTO
}
```

Ognuno dei termini presenti nel pigmento è facoltativo, ma se è presente deve essere posto nell'ordine mostrato sopra per essere sicuri che il risultato sia quello desiderato. Qualunque termine dopo l'identificatore del pigmento modifica o sovrascrive le impostazioni date nell'identificatore. Se non viene specificato alcun identificatore allora il termine modificherà i valori del pigmento nella texture predefinita. Modificatori del pigmento validi sono : `color_map`, `pigment_map`, `image_map` e `quick_color` così come ognuno dei modificatori generici come traslazione, rotazione, ridimensionamento, turbolenza, forme d'onda e warp. Questi modificatori si applicano soltanto al pigmento e non ad altre parti della texture. I modificatori devono essere specificati in fondo. I vari pattern si possono grossolanamente dividere in quattro categorie. Ogni categoria verrà discussa in seguito. Sono : pigmenti a tinte unite, pigmenti a liste di colori, mappature di colori, mappature di immagini.

7.6.1.1 Pigmento a Colore Unico

Il tipo più semplice di pigmento è costituito da un unico colore (*solid color*). Per specificare questo colore basta inserire all'interno della definizione di pigmento il colore. Per esempio :

```
pigment {color Orange}
```

la specificazione del colore è costituita dalla parola chiave `color` seguita dalla quantità di rosso, verde, blu, filtro e trasparenza della superficie. Vedere la sezione "Specificare i Colori". Qualunque modificatore dei motivi usato con un colore solido non è considerato, non essendoci alcun motivo da modificare.

7.6.1.2 Pigmento a Lista di Colori

Esistono tre motivi geometrici a lista di colore : `checker`, `hexagon` e `brick`. Il risultato sarà un motivo costituito da zone a tinta unita disposte geometricamente. Ognuno di questi motivi sarà spiegato in una sezione successiva. La sintassi è :

```
pigment { brick colore1, colore2 Modificatori ... }
pigment { checker colore1, colore2 Modificatori ... }
pigment { hexagon colore1, colore2, colore3 Modificatori ... }
```

ogni `coloreN` indica una valida specificazione di colore. Ci deve essere una virgola a separare i colori della lista oppure si dovrà usare la parola chiave `color` come separatore, cosicché POV-Ray possa determinare dove inizia e finisce ogni definizione del colore.

7.6.1.3 Mappe di Colore

La maggior parte dei motivi di colori passa bruscamente da un colore all'altro e utilizza solamente due o tre colori. Invece è possibile utilizzare dei colori che siano frutto di una transizione graduale da una tonalità ad un altra. Questo tipo di colorazione è definita in un modificatore di pigmento chiamato `color_map` che imposta il passaggio da un colore ad un altro.
Ognuno dei vari tipi di motivi che sono disponibili è caratterizzato da una funzione matematica che prende le coordinate x, y e z della posizione e le trasforma in un numero compreso tra 0.0 e 1.0 inclusi. Quel numero viene usato per specificare quale combinazione di colori è da usare in quel punto della `color_map`.

Una mappa di colore è specificata nel modo seguente :

```
pigment{
tipo_di_motivo
color_map {
[ numero_1 COLORE_1]
[ numero_2 COLORE_2]
[ numero_3 COLORE_3]
...
}
modificatori_di_pigmento...
}
```

Dove `numero_1`, `numero_2`, ... sono valori decimali compresi tra 0.0 e 1.0 questi inclusi e COLORE_1, COLORE_2, ... sono colori. Notare che le parentesi quadre "[ ]" sono parte integrante della frase, non sono notazioni che segnalano una parte facoltativa. Le parentesi definiscono ciascun valore nella mappa di colore. Ci possono essere da 2 a 256 diversi colori nella mappa. Si può anche usare la dizione alternativa `colour_map`.
Per esempio :

```
sphere {
<0,1,2>, 2
pigment {
gradient x //questo è un tipo di pigmento
color_map {
[0.1 color Red]
[0.3 color Yellow]
[0.6 color Blue]
[0.6 color Green]
[0.8 color Cyan]
}
}
}
```

qui viene calcolata la funzione del motivo e il risultato sarà un valore da 0.0 a 1.0. Se il valore è minore del primo numero, in questo caso 0.1, allora sarà usato il primo colore che è il rosso. I valori da 0.1 a 0.3 usano colori che passeranno dal rosso al giallo usando un'interpolazione lineare tra i due colori. Ugualmente i valori tra 0.3 e 0.6 useranno colori che vanno dal giallo al blu. Da notare che il terzo e il quarto numero hanno entrambi valore 0.6. Questo determinerà un passaggio immediato dal blu al verde. Più esattamente, un valore anche solo leggermente minore di 0.6 sarà blu, da 0.6 il colore sarà verde. Andando avanti, i valori tra 0.6 e 0.8 passeranno dal verde al ciano e

concludendo ogni valore maggiore di 0.8 sarà ciano.

Se non vuoi che il colore cambi in una determinata area, basterà mettere lo stesso colore in due zone adiacenti. Per esempio :

```
color_map {
[0.1 color Red]
[0.3 color Yellow]
[0.6 color Yellow]
[0.8 color Green]
}
```

in questo caso ogni valore tra 0.3 e 0.6 sarà giallo.

La parola chiave `color_map` potrà essere usata con ogni motivo eccetto `brick`, `checker`, `hexagon` e `image_map`. E' possibile dichiarare e usare identificatori della `color_map`. Per esempio:

```
#declare Colori_dell'Arcobaleno=
color_map {
[0.0 color Magenta]
[0.33 color Yellow]
[0.67 color Cyan]
[1.0 color Magenta]
}

object{Coso
pigment{
gradient x
color_map{Colori_dell'Arcobaleno}
}
}
```

7.6.1.4 Mappe di Pigmento

Oltre a poter definire `color_map`, si possono definire anche `pigment_map`. La sintassi per questo di comando è la stessa che per `color_map`, con l'eccezione che si specifica un pigmento e non un colore.
La sintassi specifica per la `pigment_map` è :

```
pigment{
TIPO_DI_MOTIVO
pigment_map {
[ NUMERO_1 PIGMENT_BODY_1]
[ NUMERO_2 PIGMENT_BODY_2]
[ NUMERO_3 PIGMENT_BODY_3]
...
}
MODIFICATORI_DEL_PIGMENTO
}
```

Dove NUMERO_1, NUMERO_2... sono numeri decimali compresi tra 0.0 e 1.0 questi inclusi. Il

PIGMENT_BODY è costituito da tutto ciò che normalmente sarebbe incluso nella sintassi di un `pigment`, ma in questo caso non è necessaria la parola chiave `pigment` e non sono necessarie neanche le parentesi graffe. E' da notare che le parentesi quadre sono parte integrante del comando. Non denotano in questo caso parti di comando facoltative. Le parentesi circondano ogni nuovo fattore che viene aggiunto alla `pigment_map`. Si possono avere da 2 a 256 fattori.
Per esempio :

```
sphere {
<0,1,2>, 2
pigment {
gradient x //questo è il TIPO_DI_MOTIVO
pigment_map {
[0.3 wood scale 0.2]
[0.3 Jade] //questo è un identificatore di pigmento
[0.6 Jade]
[0.9 marble turbulence 1]
}
}
}
```

Quando la funzione `gradient x` dà valori compresi tra 0.0 e 0.3 verrà usato il pigmento `wood` opportunamente scalato. Per valori da 0.3 a 0.6 si userà il pigmento `Jade`, da 0.6 a 0.9 si passerà gradualmente a `marble` per il quale è stata specificata una certa turbolenza e per valori superiori a 0.9 sarà usato solo i `marble`.

Le mappe di pigmenti possono essere rese complesse quanto si vuole. I pigmenti che la compongono possono avere a loro volta `pigment_map` o `color_map` o qualunque tipo di pigmento. Ogni fattore della `pigment_map` potrà essere un colore solido. Quando tutti i fattori saranno colori solidi sarebbe più opportuno, per la maggiore rapidità del rendering, che venisse usata la `color_map`.

Si possono usare anche tipi di pigmenti più complessi come quelli che coinvolgono `checker`, `hexagon` e `brick`. Per esempio...

```
pigment {
checker
pigment { Jade scale .8 }
pigment { White_Marble scale .5 }
}
```

In questi casi sono necessarie le parentesi per capire esattamente dove finisce il pigmento precedente ed inizia il successivo.
La `pigment_map` è anche usata con la *media* dei pigmenti dati. Vedere "Average (Media)" per ulteriori dettagli.

Non è possibile usare pigmenti che abbiano un'`image_map`. Vedere la sezione "Mappe di Texture" per modi alternativi da utilizzare per ottenere questo risultato.

7.6.1.5 Mappatura di un'Immagine

Quando tutto fallisce e nessuno dei pigmenti descritti qui sopra ti è di una qualche utilità per realizzare la tua immagine, puoi usare una immagine come *mappa*, per avvolgere cioè il tuo oggetto tridimensionale con un'immagine 2D.

7.6.1.5.1 Specificare un'Immagine

La sintassi per la mappatura di un'immagine (parola chiave `image_map`) è..

```
pigment {
image_map {
TIPO_DI_FILE "nome del file"
MODIFICATORI...
}
}
```

Dove `TIPO_DI_FILE` specifica uno dei seguenti formati `gif`, `tga`, `iff`, `ppm`, `pgm`, `png` o `sys`. Questo sarà seguito dal nome del file tra virgolette. Ci possono essere alcuni modificatori dopo questa specificazione. Questi modificatori sono descritti sotto. Alcune versioni precedenti di POV-Ray permettevano di inserire alcuni modificatori prima dell'opzione che specifica il tipo di file, ma questa sintassi è stata soppiantata dalla nuova, qui descritta.

Il nome del file dirà a POV-Ray quale file cercare e POV-Ray lo cercherà prima nella directory corrente e poi, se non lo ha trovato, lo cercherà nelle directory specificate nelle librerie. Questo per facilitare l'archiviazione di tutte le immagini per le mappe in una specifica sottodirectory che potrà essere eventualmente inclusa nei percorsi specificati dalle librerie.

Di default le immagini sono mappate sul piano x-y. L'immagine è proiettata sull'oggetto come se ci fosse un proiettore sull'asse z. l'immagine riempie l'area dal punto di coordinate x-y (0,0) al punto (1,1), senza considerare le dimensioni dell'immagine di partenza. Se si vuole cambiare questa posizione è necessario traslare, ruotare, ridimensionare la texture in modo da mapparla attorno all'oggetto nel modo desiderato.

Nella sezione "Checker (Scacchiera)" viene spiegato questo tipo di pigmento ed è descritto come un insieme di cubi di due colori diversi, disposti in maniera alternata, dai quali gli oggetti sono intagliati. Con l'`image_map` è la stessa cosa, dovresti immaginare che ogni punto sia una bacchetta quadrata lunga e sottile che si estende parallelamente all'asse z. L'immagine è costituita da righe e colonne di queste bacchette legate insieme in un blocco e l'oggetto viene intagliato dal blocco.

Se si vuole cambiare questa posizione è necessario traslare, ruotare, ridimensionare la texture in modo da mapparla attorno all'oggetto nel modo desiderato.

7.6.1.5.2 L'Opzione Map_Type

La proiezione dell'immagine sul piano x-y è chiamata `mappatura di tipo planare`. Questa opzione può essere cambiata aggiungendo la parola chiave `map_type` seguita da un numero che specifica il modo di mappatura.

`map_type 0` da la mappatura predefinita e quindi planare già ampiamente descritta.

`map_type 1` da una mappatura sferica. Assume che l'oggetto sia una sfera di qualunque dimensione posta nell'origine. L'asse y rappresenta il polo sud e nord della sfera. La cima e il fondo dell'immagine toccano i due poli senza tenere conto delle dimensioni precedenti dell'immagine. La

parte destra dell'immagine inizia dal lato dell'asse x positivo e trascina l'immagine lungo una rotazione intorno all'asse y. L'immagine copre la sfera esattamente una volta. La parola chiave `once` non ha nessun rapporto con questi comandi.

Con `map_type 2` si avrà una mappatura cilindrica. Si assume che un cilindro di un qualunque diametro giaccia lungo l'asse y. L'immagine si srotola intorno al cilindro con le stesse modalità della sfera e rimane alta 1 cioè da y=0 a y=1. Questo è applicato a tutte le altezze superiori a meno che non sia specificata la parola chiave `once`.

Infine `map_type 5` è a forma di toro. Si assume cioè che un toro di raggio maggiore uguale ad uno sia posizionato all'origine del piano x-z. L'immagine si comporta come nei due metodi descritti sopra. La parte superiore ed inferiore del contorno della mappa si muoveranno attorno al toro e si uniranno nella parte interna del toro.
I tipi 3 e 4 sono ancora in via di ampliamento.

Notare che le opzioni `map_type` possono essere applicate anche alla `bump_map` e alla `material_map`.

7.6.1.5.3 I Modificatori Filter e Transmit

Per rendere trasparente parte tutta la mappa si possono specificare i valori di trasparenza per la tavolozza di colori utilizzata dalle immagini PNG, GIF, IFF. Si può fare questo aggiungendo dopo il nome del file la parola chiave `filter` o `transmit`. La parola chiave è seguita da due numeri. Il primo numero rappresenta il numero corrispondente al colore della tavolozza da rendere trasparente, il secondo rappresenta la quantità di trasparenza. Questi valori dovrebbero essere separati da virgole. Per esempio :

```
image_map {
gif "mypic.gif"
filter 0, 0.5 // rendi il colore 0 trasparente al 50% (tr.
Filtrata)
filter 5, 1.0 // rendi il colore 5 trasparente al 100% (tr.
Filtrata)
transmit 8, 0.3 // rendi il colore 8 trasparente al 30% (tr. Non
filtrata)
}
```

E' possibile dare all'intera immagine un volore di `filter` o di `transmit` usando `filter all` o `transmit all` seguiti dal valore. Per esempio :

```
image_map {
gif "stnglass.gif"
filter all 0.9
}
```

Le precedenti versioni di POV-Ray usavano la parola chiave `alpha` per specificare la trasparenza, comunque questa parola viene spesso usata per descrivere la trasparenza non filtrata. Per questa ragione `alpha` non è più usato.

Vedere la sezione "Specificare i Colori" per ulteriori dettagli sulle differenze tra la trasparenza filtrata e non filtrata.

7.6.1.5.4 Usare il Canale Alfa

Un altro modo per specificare la trasparenza non filtrata in una mappa di immagine è quello di usare il canale *alfa*.
Immagini PNG permettono di memorizzare diverse trasparenze per ciascun valore di colore. Se il tuo programma per le immagini in 2D può trattare immagini PNG, allora è possibile specificare le zone di trasparenza da questo programma, piuttosto che specificare il valore di trasparenza per ogni colore da POV-Ray. dal momento che i formati PNG e TGA possono immagazzinare anche i dati riguardanti il canale alfa, puoi generare mappe di immagini che abbiano la trasparenza dipendente non dal colore del pixel, ma dalla sua posizione.

POV-Ray usa `trasmit 0.0` per non avere trasparenza e `1.0` per la totale invisibilità. I dati del canale alfa si muovono da 0 a 255 nella direzione opposta. Il valore del canale alfa 0 significa la stessa cosa che `trasmit 1.0` e il valore 255 è equivalente a `trasmit 0`.

7.6.1.6 Quick Color

Quando si sviluppano le scene di POV-Ray è spesso utile fare prove rendering di bassa qualità ma molto veloci. Il parametro `+Q` può essere usato per disattivare delle texture o dei calcoli sulle luci piuttosto impegnativi in modo da aumentare la velocità di rendering. Tutte le impostazioni di questo parametro minori di 5 non permetteranno il calcolo dei pigmenti e creeranno oggetti grigi.

Aggiungendo la parola chiave `quick_color` al pigmento è possibile usare nel rendering solamente colori uniti invece di più problematici motivi. Per esempio :

```
pigment {
gradient x
color_map{
[0.0 color Yellow]
[0.3 color Cyan]
[0.6 color Magenta]
[1.0 color Cyan]
}
turbulence 0.5
lambda 1.5
omega 0.75
octaves 8
quick_color Neon_Pink
}
```

dice a POV-Ray di usare il colore `Neon_Pink` per prove di qualità inferiore a 5, ma di usare il motivo originale per prove con qualità 6 o maggiori.

È da notare come i colori solidi come Magenta

```
pigment {color Magenta}
```

impostino immediatamente la parola chiave `quick_color` con quel valore. Puoi soprascrivere questo valore impostato automaticamente. Supposto che tu abbia dieci sfere nella scena e che siano tutte gialle, se le vuoi identificare tutte singolarmente dovrai dare ad ognuna di esse un valore di quick_color diverso, in questo modo :

```
sphere {
```

```
<1,2,3>,4
pigment { color Yellow quick_color Red }
}

sphere {
<-1,-2,-3>,4
pigment { color Yellow quick_color Blue }
}
```

e così via. A valori 6 o maggiori del parametro +Q saranno tutte gialle, ma per valori inferiori a 5 saranno diversamente colorate in modo da poterle identificare esattamente.

7.6.2 Normale

Il raytracing è conosciuto soprattutto per il modo in cui vengono rese la riflessione, la rifrazione e gli effetti luminosi. La maggior parte delle nostre percezioni dipendono dalla capacità di riflessione degli oggetti che ci circondano. Il raytracing è in grado di attuare degli scherzi alla nostra percezione in modo da creare effetti ottici e farci vedere dettagli complessi, là dove non esistono realmente.

Supposto che si desideri una superficie molto scabra, potrebbe essere molto difficile modellare matematicamente ogni rugosità. Possiamo però ricreare l'effetto intervenendo sul modo in cui la superficie si comporta nella riflessione della luce. Il calcolo della riflessione dipende da un vettore chiamato *normale alla superficie* (*surface normal*). Questo vettore parte dalla superficie ed è diretto perpendicolarmente ad essa. Manipolando artificialmente questo vettore si possono simulare superfici non lisce.

Le frasi relative alle normali sono frasi che fanno parte della sintassi delle texture e definiscono in particolare le modifiche da applicare alla componente normale della superficie. Come le frasi relative al pigmento, è possibile omettere l'intero blocco delle frasi relative alla texture più propriamente, diminuendo i tempi di battitura. Non bisogna però dimenticarsi che comunque è implicito il concetto di texture. Per esempio...

```
//questa...
object {
Coso
texture {pigment {color Purple}
normal {bumps 0.3}
}}

//può essere accorciata così...
object {
Coso
pigment {color Purple}
normal {bumps 0.3}}
```

Inserire una frase per le normali non modifica realmente le superfici. Modifica solamente il modo in cui le luci vengono rifratte e riflesse.

Il modo completo per definire le normali è questo...

```
normal {
IDENTIFICATORE_DI_NORMALI
```

```
TIPO_DI_MOTIVO valore decimale
MODIFICATORI_DI_NORMALI
TRANSFORMAZIONI...
}
```

Ognuno degli elementi che compongono l'intera sintassi delle normali è facoltativo, ma se è presente deve essere nell'ordine che è mostrato qui sopra per essere sicuri che il risultato sia quello desiderato. Ogni frase dopo l' `IDENTIFICATORE_DELLE_NORMALI` modificherà o sovrascriverà precedenti impostazioni. Se non viene specificato alcun identificatore allora la frase modificherà i valori predefiniti delle normali nella texture. Il `TIPO_DI_MOTIVO` può essere seguito da un valore decimale che controlli la profondità apparente delle irregolarità. I valori possono variare tra 0.0 e 1.0, ma possono essere usati anche altri valori. Valori negativi invertono l'effetto. Il valore predefinito, usato se non ne viene specificato un altro, sarà 0.5.

Valori validi per i modificatori delle normali sono le parole chiave `slope_map`, `normal_map`, `bump_map` e `bump_size` così come tutti i generici modificatori di motivi, quali la rotazione, la traslazione, il ridimensionamento, la turbolenza le ondulazioni e il warp. Questi modificatori verranno applicati solo alle normali e non ad altre parti della texture. I modificatori dovrebbero essere specificati per ultimi.

Ci sono tre tipi fondamentali di pattern di normali. Sono pattern, speciali e mappe bump. Si differenziano per i tipi di modificatori che si possono usare. All'inizio POV-Ray aveva dei motivi (pattern) che erano riservati alle normali. Dalla versione 3.0 di POV-Ray è possibile usare qualunque motivo sia per i pigmenti che per le normali. Per esempio è possibile usare `ripples` come pigmento o `wood` come tipo di normale. I motivi `bumps`, `dents`, `ripples`, `waves`, `wrinkles` e `bump_map` che prima erano esclusivamente motivi validi per le normali ora possono essere usati anche come pigmenti. Siccome questi sei tipi in particolare usano i calcoli relativi alle normali, non possono avere modificatori quali `slope_map`, `normal_map` o `wave_shape`. Tutti gli altri tipi di motivi possono invece utilizzarli.

7.6.2.1 Mappe Slope

`Slope_map` è un modificatore dei motivi delle normali che ti da una grande possibilità di controllo riguardo all'esatta forma delle irregolarità della superficie. E' meglio illustrato con delle normali a gradiente. Supponi di avere...

```
plane{ z, 0
pigment{ White }
normal { gradient x }
}
```

Questo dà un motivo a dente di sega che ricorda delle piccole creste che salgono dal punto di ascissa 0 al punto di ascissa 1 e poi cadono bruscamente di nuovo a zero per ricominciare a salire dal punto di ascissa 1 al punto di ascissa 2. Una `slope_map` modifica questo motivo a dente di sega in (quasi) ogni forma d'onda desiderabile. La sintassi è la seguente :

```
normal{
TIPO_DI_PATTERN Valore
slope_map {
[ NUM_1 POINT_SLOPE_1]
[ NUM_2 POINT_SLOPE_2]
[ NUM_3 POINT_SLOPE_3]
```

```
...
}
MODIFICATORI...
}
```

E' da notare che le parentesi quadre sono parti integranti della frase. Non sono simboli che denotano parti facoltative. Le parentesi delimitano ogni singolo componente della mappa. Ci possono essere da 2 a 256 componenti.
I vari `NUM_1`, `NUM_2...` sono valori decimali compresi tra 0.0 ed 1.0 inclusi. `POINT_SLOPE_1` e `POINT_SLOPE_2` sono vettori a due componenti come <0,1> dove il primo valore rappresenta l'altezza apparente dell'onda e il secondo valore rappresenta la pendenza dell'onda in quel punto. L'altezza può variare tra 0.0 ed 1.0 ma qualunque valore può essere usato. Il valore della pendenza rappresenta il cambiamento in altezza per unità di distanza (*dz/dx*). Per esempio, una pendenza di 0 significa che non c'è irregolarità nella superficie, una pendenza di 1 significa che la pendenza arriva a 45° ed una pendenza di -1 significa che la pendenza sarà di 45°, ma rivolta in senso opposto. Teoricamente, una pendenza verticale avrà un valore di inclinazione infinito. In pratica i valori di inclinazione devono variare entro un raggio che va da -3.0 a +3.0. E' bene ricordarsi che questa è solo una pendenza apparente. Un modificatore di normali non cambia la superficie dell'oggetto.
Per esempio, ecco come ottenere un onda triangolare :

```
normal {
gradient x // questo è il TIPO_DI_PATTERN
slope_map {
[0 <0, 1>] // inizia in basso e sali
[0.5 <1, 1>] // a metà continua a salire
[0.5 <1,-1>] // scendi bruscamente
[1 <0,-1>] // termina la pendenza in fondo
}
}
```

La funzione del motivo è stata calcolata e il risultato è un valore tra 0.0 e 1.0. Il primo elemento imposta che a valori x=0 l'altezza apparente sarà 1 e l'inclinazione1. A x=0.5 saremo sempre ad altezza 1 e inclinazione 1. Il terzo elemento specifica che a x=0.5 (e per alcune frazioni di poco superiori a 0.5) avremo altezza 1 e inclinazione -1. Infine a x= 1 siamo ad altezza 0 e la pendenza sarà ancora -1.

Questo esempio connette fra di loro i vari punti usando linee dritte anche se il calcolo viene effettuato per mezzo di una spline cubica. Questo esempio crea una linea ondulata.

```
normal {
gradient x // questo è il TIPO_DI_MOTIVO slope_map {
[0 <0.5, 1>] // inizia al centro e sale
[0.25 <1.0, 0>] // al centro si ha una pendenza piana
[0.5 <0.5,-1>] // dal punto centrale inizia a scendere
[0.75 <0.0, 0>] // inclinazione piatta in fondo
[1 <0.5, 1>] // finisce al centro e cresce di nuovo
}
```

L'esempio inizia all'altezza di 0.5 e cresce fino ad 1, ad un quarto della funzione siamo ad altezza 1 e inclinazione 0. Lo spazio tra questi due punti è delimitato da una curva che sale lentamente perché il valore iniziale e finale della pendenza sono diversi. A metà siamo a mezza altezza, la curva sta

scendendo e arriverà in fondo a ¾ del suo percorso. Bisognerà arrivare fino al valore 1 per completare il ciclo. Ci sono molti esempi nella scena di esempio **slopemap.pov**.

Una `slope_map` può usare qualunque motivo eccetto `brick`, `checker`, `hexagon`, `bumps`, `dents`, `ripples`, `waves`, `wrinkles` e `bump_map`.

Si possono dichiarare e poi usare identificatori di `slope_map`. Per esempio :

```
#declare Fancy_Wave =
slope_map { // ora iniziamo a inventare
[0.0 <0, 1>] // fai un triangolino
[0.2 <1, 1>] // giù
[0.2 <1,-1>] // fino a
[0.4 <0,-1>] // qui.
[0.4 <0, 0>] // area piatta
[0.5 <0, 0>] // passando da qui.
[0.5 <1, 0>] // primo spigolo dell'onda quadra
[0.6 <1, 0>] // secondo spigolo
[0.6 <0, 0>] // di nuovo piatto
[0.7 <0, 0>] // qui.
[0.7 <0, 3>] // inizia a salire
[0.8 <1, 0>] // piano in cima
[0.9 <0,-3>] // qui finisce.
[0.9 <0, 0>] // il resto è piano fino a 1.0
}

object{ Coso
pigment { White }
normal {
wood
slope_map { Fancy_Wave }
}
}
```

7.6.2.2 Mappe di Normali

La maggior parte delle volte applicherai un motivo con una normale semplice ad un'intera superficie, ma puoi anche creare un motivo od un insieme di normali usando una mappatura delle normali. La sintassi per la mappatura delle normali è uguale alla sintassi usata per quella dei pigmenti tranne per il fatto che è necessario specificare una normale per ogni componente.
Una mappatura delle normali si specifica così :

```
normal{
TIPO_DI_PATTERN
normal_map {
[ NUM_1 NORMALE_1]
[ NUM_2 NORMALE_2]
[ NUM_3 NORMALE_3]
...
}
MODIFICATORI...
}
```

Dove `NUM_1`, `NUM_2`... sono valori decimali compresi tra 0.0 e 1.0 inclusi e `NORMALE_N` è qualunque cosa che di solito apparirebbe all'interno di una frase relativa alle normali, ma le usuali parole chiave e le parentesi graffe non sono necessarie. Le parentesi quadre sono invece parte integrante della frase. Non sono notazioni che indicano parti facoltative. Le parentesi definiscono ogni componente. Ci possono essere da 2 a 256 componenti nella mappa.
Per esempio,

```
normal {
gradient x //questo è il TIPO_DI_MOTIVO
normal_map {
[0.3 bumps scale 2]
[0.3 dents]
[0.6 dents]
[0.9 marble turbulence 1]
}
}
```

Quando la funzione `gradient x` assume valori compresi tra 0.0 e 0.3 allora è usata una normale `bumps` scalata. Da 0.3 a 0.6 vale la normale `dents`, da 0.6 a 0.9 si passa dalla normale `dents` a `marble` con turbolenza. Da 0.9 in poi viene usato solo `marble`.

Le mappe delle normali possono essere complesse quanto si vuole. Le normali presenti in una mappa possono avere `slope_maps` o `normal_maps` di ogni tipo.
La mappatura delle normali è usata con la parola chiave `average`. Vedi "Average (Media)" per ulteriori informazioni.
Le normali possono anche essere usate con motivi come : `checker`, `hexagon` e `brick`. Per esempio...

```
normal {
checker
normal { gradient x scale .2 }
normal { gradient y scale .2 }
}
}
```

E' da notare che in casi di motivi come quelli appena elencati, la frase `normal{...}` deve precedere il pattern.
Non puoi usare `normal_map` o singole normali con una `bump_map`. Vedere il paragrafo "Mappe di Texture" per soluzioni alternative.

7.6.2.3 Mappe Bump

Quando non funziona niente e nessun tipo di normale già descritto ti soddisfa, puoi usare una `bump_map` , una immagine come mappa, per mettere cioè intorno al tuo oggetto tridimensionale un'immagine 2D che ne simula le irregolarità della superficie.

Invece di porre i colori dell'immagine sull'oggetto come per l'`image_map`, la `bump_map` modifica apparentemente la superficie basandosi sul colore dell'immagine in quel punto. Il risultato sarà simile ad un'immagine che è stata incisa sulla superficie. Una `bump_map` usa di default la luminosità dell'immagine. I colori sono convertiti a toni di grigio internamente prima di calcolarne l'altezza. Il nero risulta in un avvallamento, il bianco in un rilievo.

Alternativamente si possono usare i valori numerici indicati nella tavolozza di colori dell'immagine (vedi il paragrafo "Use_Index e Use_Color".

7.6.2.3.1 Specificare una Mappa Bump

La sintassi per una bump_map è...

```
normal {
bump_map {
TIPO_DI_FILE "nomefile.est"
MODIFICATORI_DELL'IMMAGINE...
}
MODIFICATORI_DELLE_NORMALI...
}
```

dove TIPO_DI_FILE è una delle seguenti parole chiave : gif, tga, iff, ppm, pgm, png o sys. Questa viene seguita dal nome del file scritto usando qualunque stringa valida. Alcuni modificatori facoltativi possono seguire la specificazione del file. I modificatori sono descritti più sotto. E' da notare che le prime versioni di POV-Ray permettevano alcuni modificatori prima di TIPO_DI_FILE, ma questo tipo di sintassi è stato sostituito in favore di quella descritta sopra.
I nomi dei file specificati in una frase bump_map{...}saranno cercati nella directory corrente e poi, se non sono stati trovati, nelle directory specificate dal parametro +L o dall'opzione Library_Path. Questo tende a semplificare la possibilità di tenere tutti i file di bump_map in una sottodirectory separata e quindi specificare un percorso per essa. Per alcuni sistemi operativi il file non viene cercato nei percorsi predefiniti a meno che tu non specifichi questi percorsi come Library_Path. La bump_map viene mappata sul piano x-y. I rilievi vengono proiettati sull'oggetto come se ci fosse un proiettore sull'asse z. Il rilievo copre interamente l'area quadrata compresa tra le coordinate <0,0> e <1,1>, senza tenere conto delle dimensioni originarie dell'immagine. Se vuoi cambiare questo valore predefinito puoi traslare, ruotare o scalare le normali o le texture per mapparle sulla superficie dell'oggetto come più ti aggrada.
Il nome del file è opzionalmente seguito da uno o più modificatori dell'immagine. I modificatori bump_size, use_color e use_index sono specifici delle bump_map e saranno discusse nei paragrafi seguenti. Vedi il paragrafo "Modificatori di Immagine" per altri modificatori.
Dopo una frase bump_map, ma sempre all'interno delle frasi del blocco normal{...} è possibile applicare qualunque modificatore delle normali eccetto slope_map e i pattern a forma d'onda.

7.6.2.3.2 Il Parametro Bump_Size

L'ampiezza dei rilievi può essere scalata usando il modificatore bump_size. Il numero che segue questo modificatore può essere un qualunque numero diverso da zero. Valori tipici variano tra 0.1 e 4.0 o 5. 0.

```
normal {
bump_map {
gif "stuff.gif"
bump_size 5.0
}
}
```

Prima, questo modificatore poteva essere usato solo all'interno di una mappatura dei rilievi. Ora può essere usato con ogni normale. Bump_size è usato prevalentemente per sovrascrivere valori

definiti in precedenza. Per esempio,

```
normal {
My_Normal //questo è un identificatore già definito
bump_size 2.0
}
}
```

7.6.2.3.3 Use_Index e Use_Color

Di solito la mappatura dei rilievi converte il colore dei pixel nella mappa in scala di grigi, con valori compresi nell'intervallo 0.0 1.0 e calcola l'altezza dei rilievi basandosi su quei valori. Se viene specificato `use_index` la mappatura dei rilievi userà invece il numero progressivo che i colori hanno nella tavolozza. Quindi, il colore 0 sarà in basso, mentre il colore 255 (se l'immagine ha 256 colori) darà i rilievi più alti. In questo calcolo non viene preso in considerazione il colore effettivo del pixel. Questa opzione è disponibile solo su quei formati che si basano sulla tavolozza di colori. La parola chiave `use_color` può essere usata per specificare esplicitamente che si devono invece usare i colori. E' valida anche la dizione alternativa `use_colour`. Questi modificatori possono essere usati esclusivamente all'interno di frasi relative a mappature dei rilievi.

7.6.3 Finitura

Le proprietà della finitura (*finish*) di una superficie possono influenzare notevolmente il suo aspetto. Come si riflette la luce sull'oggetto ? Cosa succede quando la luce lo attraversa ? Per rispondere a queste domande è necessario saperne di più sulle frasi che regolano la finitura dell'oggetto.
Anche queste frasi fanno parte delle impostazioni della texture e definiscono varie proprietà della finitura da applicare ad un oggetto. Come per le frasi riguardanti i pigmenti e le normali è possibile evitare buona parte della sintassi relativa alle texture per passare direttamente a quella relativa alla finitura. Non bisogna però dimenticare che l'uso di una finitura implica la presenza di una texture anche se non è stata esplicitamente citata. Per esempio :

```
//Questo
object{
Coso
texture{
pigment {color Purple}
finish{phong 0.3}
}
}

//può essere abbreviato così...
object{
Coso
pigment {color Purple}
finish{phong 0.3}
}
```

la forma completa per definire le finiture è...

```
finish {
IDENTIFICATORE_DI_FINITURA
[ ambient COLORE ]
[ diffuse NUMERO DECIMALE ]
```

```
[ brilliance NUMERO DECIMALE ]
[ phong NUMERO DECIMALE ]
[ phong_size NUMERO DECIMALE ]
[ specular NUMERO DECIMALE ]
[ roughness NUMERO DECIMALE ]
[ metallic [NUMERO DECIMALE ] ]
[ reflection COLORE ]
[ refraction NUMERO DECIMALE ]
[ ior NUMERO DECIMALE ]
[ caustics NUMERO DECIMALE ]
[ fade_distance NUMERO DECIMALE ]
[ fade_power NUMERO DECIMALE ]
[ irid { thickness NUMERO DECIMALE turbulence VETTORI } ]
[ crand NUMERO DECIMALE ]
}
```

L'`IDENTIFICATORE_DI_FINITURA` è facoltativo, ma se è presente deve precedere tutti le altre frasi. Qualunque frase che lo segue può sostituire le impostazioni lì definite. Se non viene specificato alcun identificatore allora le frasi che seguono modificheranno i valori della finitura predefinita. Non sono permesse le trasformazioni all'interno della finitura, poiché questa ricopre interamente la superficie.

7.6.3.1 Luce Ambiente

La luce che si vede nelle parti in ombra proviene dalla riflessione di luce diffusa di altri oggetti. Questa luce non può essere modellata direttamente. Possiamo comunque usare un trucco chiamato `luce ambient` per simulare la luce delle zone in ombra.

La luce ambiente è una luce che è diffusa ovunque nella scena. E' presente dappertutto e illumina le parti in ombra degli oggetti. Calcolare l'esatta quantità di luce ambiente nella scena sarebbe troppo lungo, così è possibile simulare la luce ambiente aggiungendo una piccola quantità di luce bianca a ogni texture sia dove l'oggetto è illuminato sia dove è in ombra.
Questo significa che le porzioni di oggetto che saranno completamente in ombra, conserveranno comunque un po' del loro colore. E' un po' come se l'oggetto passasse dall'illuminazione con la luce ambiente alla vera e propria texture.
Anche se la sintassi richiederebbe un colore, al momento è necessario specificare soltanto un valore decimale. Per esempio il valore 0.3 creerà il vettore colore <0.3,0.3,0.3,0.3,0.3> che è accettabile perché sono usate solo le componenti rosse, verdi e blu.
Il valore predefinito è molto scarso, solo 0.1. Il valore può variare tra 0.0 e 1.0. La luce ambiente investirà le parti illuminate e quelle in ombra, se alzi il valore della luce ambiente è probabile che tu desideri abbassare quello della luce diffusa.

Questo metodo non tiene conto della luce riflessa dagli oggetti circostanti. Se si cammina in una stanza che ha le pareti, il pavimento ed il soffitto rosso si potrà notare come i nostri abiti bianchi assumano una colorazione rosa determinata dalla luce riflessa. La luce ambiente di POV-Ray non tiene conto di questo. Parimenti non c'è modo di tenere conto della luce indiretta, speculare e riflessa, come una luce che viene riflessa da uno specchio.

E' possibile colorare la luce ambiente usando uno di questi due metodi. Puoi specificare un colore con un valore decimale dopo la parola chiave `ambient`, in ognuna delle frasi relative alla finitura. Per esempio :

```
finish { ambient rgb <0.3,0.1,0.1> } //luce ambiente rosa
```

Puoi anche specificare la luce ambiente usata per calcolare la luminosità di un ambiente di un oggetto usando la frase delle impostazioni generali chiamata `ambient_light`. La formula che sarà usata nei calcoli è la seguente :

```
LUCE AMBIENTE = LUCE AMBIENTE DELLA FINITURA * LUCE AMBIENTE DELLA
SCENA
```

vedere il paragrafo "Luce Ambiente".

7.6.3.2 Riflessione Diffusa

Quando la luce si riflette su una superficie, le leggi della fisica dicono che dovrebbe lasciare la superficie con lo stesso angolo col quale l'ha raggiunta (angolo di incidenza). Questo è abbastanza simile a quello che succede quando una palla da biliardo colpisce una sponda del tavolo (esattamente uguale, N.d.T.). Questa riflessione perfetta è chiamata *riflessione speculare*. Comunque, solo superfici molto lisce riflettono in questo modo. La maggior parte delle volte, la luce viene riflessa e dispersa in tutte le direzioni dalle microscopiche asperità della superficie. Questa dispersione è chiamata *riflessione diffusa* perché la luce si diffonde in molte direzioni. E' responsabile della maggior parte della luce riflessa che vediamo.

POV-Ray e molti altri programmi di raytracing possono solo simulare uno di questi tre tipi di illuminazione : l'illuminazione diretta data dalla luce proveniente da sorgenti luminose, la luce proveniente da altri oggetti come gli specchi a causa della riflessione speculare ed infine la luce proveniente da altri oggetti a causa della riflessione diffusa (ad esempio, guardando in un angolo sotto al tavolo, non illuminato direttamente, è comunque possibile vedere qualcosa perché la luce proviene dalla riflessione diffusa degli oggetti vicini).

7.6.3.2.1 Luce Diffusa

La parola chiave `diffuse` è usata in una frase riguardante le finiture per controllare quanta parte della luce che proviene direttamente da una sorgente luminosa è riflessa grazie alla riflessione diffusa. Per esempio,

```
finish {diffuse 0.7}
```

significa che il 70% della luce visibile proviene direttamente dall'illuminazione delle sorgenti luminose. Il valore predefinito è 0.6.

7.6.3.2.2 Brillantezza

La quantità di luce diretta che viene diffusa da un oggetto dipende dall'angolo di incidenza. Quando la luce colpisce di taglio la superficie, la illumina meno, quando le si trova direttamente sopra, la illumina di più. La parola chiave `brilliance` può essere usata in una frase `finish{...}` per variare il modo in cui la luce si attenua in funzione dell'angolo di incidenza. Ciò controlla l'ampiezza dell'illuminazione diffusa degli oggetti e modifica leggermente l'aspetto della superficie. Gli oggetti possono sembrare più metallici aumentando la loro brillantezza. Il valore predefinito è 1.0. Valori più alti di 10.0 ridurranno l'attenuarsi della luce per angoli di incidenza medi e bassi. Non ci sono limiti al valore che si può assegnare alla brillantezza. Provate a vedere quale si adatta meglio alla vostra scena. E' meglio usare `brilliance` unitamente alle funzioni `phong` ecc.

7.6.3.2.3 Crand

Superfici molo ruvide, come il cemento e la sabbia, mostrano una scura granulosità nel loro colore apparente. Questo è determinato dalle ombre degli avvallamenti e dei rilievi nella superficie. La parola chiave `crand` può essere aggiunta per determinare una leggera ombreggiatura della riflessione diffusa della luce diretta. Valori tipici vanno da 0.01 a 0.5 o maggiori. Il valore predefinito è zero. Ad esempio :

```
finish { crand 0.05 }
```

La granulosità introdotta da questa funzione è applicata punto per punto. Questo significa che apparirà uguale sia in oggetti vicini che in oggetti lontani. L'effetto varierà invece in relazione alla risoluzione usata per il rendering. Per questa ragione non è un modo molto realistico di rendere le superfici granulose, ma alcuni oggetti potrebbero risultare migliori. `Crand` non dovrebbe essere usato nelle animazioni. Questa è una delle pochissime funzioni completamente casuali di POV-Ray e nelle animazioni produrrebbe delle differenze non volute tra fotogramma e fotogramma.

7.6.3.3 Punti Illuminati

Si chiamano *highlights* quei punti chiari che appaiono quando una sorgente luminosa è riflessa da un oggetto liscio. Sono dati da un misto di riflessione speculare e diffusa. Si comportano come la luce speculare perché dipendono dagli angoli di illuminazione e di osservazione. D'altra parte sono simili anche alla luce diffusa perché avviene dispersione. Per modellare con esattezza questi punti illuminati dovresti calcolare la riflessione speculare di migliaia di microscopici rilievi chiamati microsuperfici. Quanto più queste sono rivolte verso l'osservatore, più l'oggetto sembrerà luminoso e più stretta apparirà la zona del riflesso. POV-Ray usa due modelli diversi per simulare questi punti illuminati senza calcolare le microsuperfici. Questi metodi sono la riflessione speculare e la riflessione di Phong. Essi non si escludono. E' possibile specificare entrambi nello stesso oggetto, ma normalmente se ne usa solo uno.

7.6.3.3.1 Punti Illuminati con riflessione di Phong

La parola chiave `phong` controlla la quantità di riflessione di Phong. Sull'oggetto si formeranno punti luminosi del colore della sorgente luminosa che viene riflessa.
La riflessione di Phong misura l'illuminazione media delle microsuperfici rivolte verso l'osservatore. Valori tipici assegnati a questa parola chiave sono compresi nell'intervallo da 0.0 a 1.0. 1.0 determinerà la saturazione completa del colore della sorgente luminosa al centro dell'area più luminosa. Il valore predefinito è 0.0.
La dimensione del punto illuminato è definita dal valore `Phong_size`. Più grande è il valore assegnato a questa parola chiave più piccolo e più brillante sarà il punto. A valori più piccoli della parola chiave corrisponderà invece un punto più grande e una superficie meno lucente. Valori tipici per questo parametro varieranno tra 1.0 (molto opaco) a 250 (molto luminoso). Il valore predefinito per la parola chiave `phong_size` è 40 (la superficie avrà un aspetto simile alla plastica). Per esempio:

```
finish { phong 0.9 phong_size 60 }
```

se non è specificato phong la parola chiave phong_size non ha effetto.

7.6.3.3.2 Punti Illuminati con Riflessione Speculare

Un punto illuminato con riflessione speculare è molto simile ad un punto illuminato con riflessione Phong, ma nel calcolo si fa uso di metodi diversi. Il metodo speculare rende più fedelmente la riflessione reale ed è molto più credibile per la diffusione della luce del punto luminoso nei contorni.
Il valore di `specular` varia solitamente tra 0.0 e 1.0. 1.0 determinerà la saturazione completa del

colore della sorgente luminosa al centro dell'area più luminosa. Il valore predefinito è 0.0.

La dimensione del punto illuminato è definita dal valore `roughness`. Valori tipici per questo parametro varieranno tra 1.0 (superficie molto ruvida, quindi un punto molto grande) a 0.0005 (superficie molto liscia e un punto molto piccolo). Il valore predefinito per la parola chiave `roughness` è 0.05 (la superficie avrà un'apparenza simile alla plastica).

Usando questo parametro è possibile incorrere in valori errati che provocheranno un errore al momento del rendering. Non usate 0 e se incorrete in errori controllate se il valore impostato non sia troppo piccolo rispetto al valore minimo possibile. Per esempio :

```
finish {specular 0.9 roughness 0.02}
```

Se la parola chiave `specular` non è stata specificata, la parola chiave `roughness` non avrà effetto.

7.6.3.3.3 Il Modificatore Metallic

La parola chiave `metallic` può essere usata sia per i riflessi calcolati col metodo Phong sia con quello di riflessione speculare. Questa parola chiave indica il modo in cui verrà calcolato il colore del punto luminoso. In particolare questo colore sarà calcolato mediante un algoritmo empirico che tenta di simulare le capacità di riflessione di una superficie metallica.

Quando la luce è riflessa specularmente da una superficie metallica, prenderà il colore della superficie, tranne quando l'angolo di incidenza si avvicinerà a 90°. In questo caso il colore sarà bianco.

La parola chiave `metallic` può essere seguita da un numero decimale che indicherà la quantità di questo effetto, il valore predefinito è 1. Per esempio :

```
finish {
phong 0.9
phong_size 60
metallic
}
```

se non si è specificato `phong` o `specular` la parola chiave `metallic` non avrà effetto.

7.6.3.4 Riflessione Speculare

Si chiama riflessione speculare, il fenomeno per il quale la luce non si diffonde su una superficie e l'angolo di incidenza è esattamente uguale all'angolo di riflessione. Questo tipo di riflessioni è controllato da una parola chiave appartenente alle frasi delle finiture. Questa parola chiave è `reflection`. Per esempio :

```
finish { reflection 1.0 ambient 0 diffuse 0 }
```

Questa impostazione creerà uno specchio, che rifletterà tutti gli altri elementi della scena. Solitamente viene specificato un solo valore decimale dopo la parola chiave, anche se la forma sintattica completa prevederebbe anche l'impostazione di un colore. Per esempio un valore decimale come 0.3 produrrà in vettore <0.3,0.3,0.3,0.3,0.3> che è accettabile, poiché saranno prese in considerazione solo le parti riguardanti i colori rosse, verde e blu.
Il valore può variare tra 0.0 e 1.0. Di default non c'è riflessione.

Aggiungere la riflessione ad un oggetto aumenterà il tempo di rendering perché sarà necessario tracciare più raggi. La luce riflessa può essere colorata con un colore specificandolo al posto del valore decimale. Per esempio

```
finish { reflection rgb <1,0,0> }
```

creerà uno specchio che rifletterà solo il rosso.

Anche se questo tipo di riflessione si chiama `specular` non è controllata dalla parola chiave `specular`. Questa parola chiave regola solo i punti illuminati da riflessione speculare.

7.6.3.5 Rifrazione

La rifrazione è la deviazione che la luce subisce al passaggio all'interno di un corpo con una determinata densità. Normalmente in POV-Ray le superfici totalmente trasparenti o trasparenti solo in parte non rifrangono la luce. La rifrazione si attiva aggiungendo alle frasi della finitura la parola chiave `refraction 1.0`.

E' raccomandabile usare per questa parola chiave solo i valori 0 e 1 (o meglio ancora, `on` e `off`). I valori compresi all'interno di questo intervallo modificheranno il percorso della luce in modi che nulla hanno a che vedere con la rifrazione reale. Molte scene di POV-Ray sono state create con valori compresi in questo intervallo prima che fosse stato scoperto questo errore, così è stata mantenuta questa possibilità. Un modo più appropriato per ridurre la luminosità dei raggi rifratti è quello di cambiarne i valori di trasparenza e di filtro. Applicare la rifrazione ad un oggetto non vuol dire renderlo trasparente. La trasparenza sia ha solo quando i valori di `filter` o `transmit` sono diversi da zero.

La quantità di luce deviata o rifratta dipende dalla densità del materiale. Aria, acqua, cristalli e diamante hanno tutti diverse densità e quindi rifrangeranno la luce in maniere diverse. L'indice di rifrazione o `ior` viene usato dagli scienziati per descrivere la densità relativa delle sostanze. La parola chiave `ior` è usata da POV-Ray per specificare questo valore. Per esempio :

```
texture {
pigment { White filter 0.9 }
finish {
refraction 1
ior 1.5
}
}
```

Il valore di default di `ior` è di 1 e non darà luogo a rifrazione. L'indice di rifrazione dell'aria è 1.0, quello dell'acqua di 1.33, 1.5 per il vetro e quello del diamante è 2.4. Il file **consts.inc** contiene alcuni valori utili di `ior`.
Se la texture ha un filtro e non sono stati specificati valori per la rifrazione e per lo `ior`, il risultato non sarà influenzato dalla rifrazione e i raggi non saranno quindi deviati. Nel caso di texture stratificate, le parole chiave riguardanti la rifrazione e l'indice di rifrazione dovranno essere incluse nell'ultima texture, altrimenti non avranno effetto.

7.6.3.5.1 Attenuazione della Luce

L'attenuazione della luce è usata per simulare la diminuzione dell'intensità luminosa quando la luce passa attraverso un oggetto trasparente. La sua sintassi è :

```
finish {
fade_distance DISTANZA_DI_DECADIMENTO
fade_power POTENZA_DI_DECADIMENTO
}
```

La parola chiave `fade_distance` determina la distanza che la luce deve percorrere per raggiungere la metà della sua intensità iniziale, mentre la parola chiave `fade_power` determina la velocità di decadimento. Per effetti realistici, si dovrà utilizzare un valore di `fade_power` di 1 o 2. L'attenuazione è calcolata con una formula simile a quella usata per l'attenuazione delle sorgenti luminose.

$$attenuazione = \frac{1}{1 + \left(d/distanza_di_decadimento\right)^{potenza_di_decadimento}}$$

7.6.3.5.2 Falsi Riflessi

I falsi riflessi (caustics) sono effetti di luce che si verificano quando la luce è riflessa o rifratta da una superficie. Immagina un bicchiere d'acqua su un tavolo. Se la luce del Sole colpisce il vetro, si vedranno dei punti di luce sul tavolo. Alcuni di questi punti sono causati dalla riflessione del vetro, mentre altri sono causati dalla luce che viene rifratta dall'acqua nel bicchiere.
Poiché calcolare realmente questi effetti è molto difficile (se non impossibile) e dispendioso in termini di tempo, POV-Ray usa un metodo piuttosto semplice per simulare i falsi riflessi causati dalla rifrazione. Questi riflessi sono limitati all'ombra dell'oggetto trasparente. I falsi riflessi dovuti a riflessione anziché a rifrazione non verranno simulati e non si avranno falsi riflessi sulle aree che non sono in ombra.
La sintassi è :

```
finish {
caustics QUANTITA'
}
```

7.6.3.6 Iridescenza

L'iridescenza, o 'interferenza da strato sottile di Newton', simula l'effetto di una luce su una superficie coperta da una pellicola trasparente di spessore microscopico. L'effetto è simile a quello causato da una macchia d'olio su una pozza d'acqua, o ai colori che si vedono su una bolla di sapone. Vedi anche "Irid_Wavelenght".
La sintassi è :

```
finish {
irid {
QUANTITA'
thickness DECIMALE
turbulence VETTORE
}
}
```

Questa finitura modifica il colore della superficie in relazione all'angolo tra la luce e la superficie stessa. L'effetto funziona relativamente alla posizione ed all'angolo della sorgente luminosa rispetto alla superficie e non si comporta nello stesso modo di un pigmento procedurale.
Il parametro `QUANTITA'` determina il contributo dell'effetto al cambiamento di colore della superficie. Come regola generale, dovrebbe essere intorno a 0.25 (25%) o meno, ma è meglio

sperimentare. Se la superficie diventa troppo bianca, provate ad abbassare i valori della luce diffusa e della luce ambiente della superficie. La parola chiave `thickness` determina lo spessore della pellicola. Questo parametro è difficile da impostare poiché il valore dello spessore non ha alcuna relazione con le dimensioni effettive dell'oggetto. Cambiandolo si modificherà la complessità dell'effetto. Una pellicola molto sottile avrà una maggiore frequenza di cambiamento del colore, mentre una pellicola più spessa presenterà aree di colore più grandi ed uniformi.
Lo spessore della pellicola può essere cambiato con la parola chiave `turbulence`. Con l'iridescenza, si può specificare solo la quantità della turbolenza. I valori di `octaves`, `lambda` e `omega` sono predefiniti e non sono modificabili.
In più, perturbare le normali alla superficie dell'oggetto usando delle normali, modificherà l'iridescenza.
Per i curiosi, questo effetto avviene poiché quando un raggio colpisce la superficie della pellicola, parte della luce è riflessa dalla superficie, mentre una parte è trasmessa attraverso essa. Questi raggi passeranno attraverso il film e si rifletteranno contro lo strato sottostante. La luce emerge dalla pellicola leggermente fuori fase rispetto ai raggi riflessi dalla superficie.
Questo spostamento di fase crea l'interferenza, che varia con la lunghezza d'onda delle componenti dei colori. Il risultato sarà che alcune lunghezze d'onda saranno rinforzate, mentre alcune verranno annullate. Quando tutte queste componenti si ricombinano fra loro, si avrà l'iridescenza. Per implementare questa funzione si è usato come riferimento il libro *"Fundamentals of Three-Dimensional Computer Graphics" di Alan Watt (Addison-Wesley).*

7.6.4 Aloni

Informazione importante : le halo nella versione 3.0 di POV-Ray sono oggetti sperimentali. E' molto probabile che verranno cambiate nelle versioni future. Non è possibile garantire che scene che usano questa funzione rimangano identiche nel futuro o che rimanga in uso la stessa sintassi.
Un alone è utilizzato per simulare gli effetti atmosferici che accadono quando singole particelle interagiscono con la luce o emettono luce. Questi effetti includono le nuvole, la nebbia, il fuoco ecc.
L'alone è contenuto e riempie completamente un oggetto, chiamato oggetto contenitore. Se l'oggetto è parzialmente o completamente trasparente ed è cavo (*hollow*), vedere la sezione "Hollow (Vuoto)" l'alone sarà visibile. Gli effetti creati dall'alone saranno comunque limitati al volume occupato dall'oggetto. Bisogna sempre ricordarsi questo.

L'aspetto dell'alone coinvolge numerosi parametri. Primo, è necessario specificare quale tipo di oggetto si vuole simulare. Dopo di che sarà necessario chiarire la distribuzione delle particelle. E' possibile fare questo in due passaggi : selezionare una funzione di mapping e una funzione che indichi la densità. La prima funzione mapperà le coordinate del mondo in un intervallo avente una dimensione, mentre l'altra funzione definirà il modo in cui questo mappa lineare sarà applicata ai valori di densità.
Le proprietà delle particelle, come il loro colore, la loro trasparenza, sono determinate da una mappatura dei colori. I valori della densità sono calcolati dal processo di mappatura e verranno usati per determinare il colore appropriato.
Il percorso del raggio è usato sia per campionare il volume dell'alone sia per valutare l'accumulazione dell'intensità e dell'opacità di ogni campione.
I paragrafi che seguono spiegheranno in dettaglio tutte le parola chiave che possono modificare gli aloni. La sintassi completa è :

```
halo {
attenuating | emitting | glowing | dust
[ constant | linear | cubic | poly ]
[ planar_mapping | spherical_mapping |cylindrical_mapping |
box_mapping ]
```

```
[ dust_type TIPO_DI_POLVERE ]
[ eccentricity ECCENTRICITA' ]
[ max_value MAX_VALUE ]
[ exponent ESPONENTE ]
[ samples CAMPIONI ]
[ aa_level LIVELLO_DI_AA ]
[ aa_threshold SOGLIA_DI_AA ]
[ jitter JITTER ]
[ turbulence <TURBOLENZA> ]
[ octaves OTTAVE ]
[ omega OMEGA ]
[ lambda LAMBDA ]
[ colour_map MAPPATURA_DI_COLORE ]
[ frequency FREQUENZA ]
[ phase FASE ]
[ scale <VETTORE> ]
[ rotate <VETTORE> ]
[ translate <VETTORE> ]
}
```

7.6.4.1 Mappatura dell'Alone

Le variazioni dell'alone sono moltissime come è facile capire dalla lunga lista qui sopra. Qui intanto spieghiamo i passi che vengono fatti durante la creazione dell'alone ; in questa lista è anche evidenziato dove le varie parole chiave intervengono nel corso dei calcoli.

1.Viene calcolata la posizione del punto P (di coordinate x, y, z), che si trova all'interno dell'oggetto contenitore, in relazione alla posizione corrente del campionamento eseguito dal raggio. La posizione è influenzata dalla parola chiave `jitter`, dal numero di campioni e dall'uso dell'anti-aliasing (`aa_level` e `aa_threshold`).
2.Il punto P è trasformato nel punto Q usando le trasformazioni a cui è soggetto l'alone. In questa fase vengono calcolate tutte le trasformazioni specificate nelle frasi dell'alone.
3.Viene aggiunta la turbolenza al punto Q. la quantità di turbolenza è determinata dalla parola chiave `turbulence`. Il calcolo della turbolenza è influenzata dalle parola chiave `octaves`, `omega` e `lambda`.
4.Viene calcolato il raggio r in relazione alla densità propria della mappatura applicata (`planar_mapping`, `spherical_mapping`, `cylindrical_mapping`, `box_mapping`). Il raggio è tenuto nell'intervallo tra 0 e 1, cioè 0 <= r <= 1.
5.la densità d è calcolata a partire dal raggio r usando la funzione densità specificata (constante, lineare, cubica, polinomiale). Il valore massimo è determinato dalla parola chiave `max_value`. La densità varierà tra 0 e `max_value`.
6.La densità d è prima moltiplicata per il valore della frequenza, aggiunta al valore della fase e normalizzata tra 0 e 1 prima che sia usato per calcolarne il colore specificato dalla parola chiave `color_map`. Se si sta usando un'alone attenuante il colore sarà determinato dalla densità totale lungo il raggio e non dalla somma dei colori di ogni campione.

Tutti i passaggi sono ripetuti per ogni punto campione lungo il raggio che è all'interno dell'oggetto contenitore. I passaggi da 2 a 6 sono ripetuti per gli aloni uniti all'oggetto contenitore.

7.6.4.2 Aloni Multipli

E' possibile inserire più di un alone dentro ad un oggetto contenitore, semplicemente mettendo più di una frase `halo{...}` dentro la descrizione dell'oggetto contenitore, come :

```
sphere { 0, 1
pigment { Clear }
halo { la halo n. 1 }
halo { la halo n. 2 }
halo { la halo n. 3 }
...
}
```

Gli effetti di halo di tipo diverso sono sommati, in maniera simile all'operazione di unione CSG. Si deve tenere presente che halo attenuanti multiple useranno la mappatura del colore dell'ultima halo scritta. Non è possibile utilizzare mappature di colore diverse per halo attenuanti multiple.

7.6.4.3 Tipi di Alone

I tipi dell'alone sono definiti da una delle seguenti parole chiave che si escludono a vicenda (se ne viene specificata più di una, verrà usata l'ultima). La parola chiave predefinita è `attenuating`

```
halo {
attenuating | emitting | glowing | dust
}
```

Il tipo di halo determina il modo in cui la luce interagisce con le particelle all'interno dell'oggetto contenitore. Ci sono due tipi di interazione : auto-illuminata ed illuminata. Il primo tipo include gli effetti `attenuating`, `emitting` e `glowing` mentre l'effetto `dust` appartiene al secondo tipo. I quattro tipi saranno spiegati dettagliatamente nei prossimi paragrafi.

7.6.4.3.1 Alone Attenuante

L'alone attenuante assorbe solamente la luce che gli passa attraverso ed è reso accumulando la densità delle particelle lungo un raggio. Il colore totale dell'alone è determinato dal totale tra la densità accumulata e la mappatura dei colori specificata (vedi il paragrafo "Mappa del Colore per gli Aloni"). La luce dello sfondo, cioè la luce che passa attraverso la halo, è attenuata dalla densità totale, d aggiunta al colore totale della halo per determinare il colore finale.
Questo modello è usato per determinare la distribuzione delle particelle con una alta albedo, perché il colore finale non è in relazione alla trasparenza di elementi di volume, ma alla densità totale lungo un raggio (non viene calcolato un integrale sul volume occupato, ma un integrale di linea lungo il raggio di luce, N.d.T.). L'albedo di una particella è determinata dalla quantità di luce diffusa dalla particella in tutte le direzioni in relazione alla quantità di luce incidente. Se la particella non assorbe alcuna luce, l'albedo è 1. Nuvole e vapori sono due degli effetti che si possono creare un modo abbastanza realistico, aggiungendo sufficiente turbolenza.

7.6.4.3.2 Polvere

La polvere è composta da particelle che non emettono luce. Essa si limita ad riflettere e assorbire la luce incidente. La sua sintassi è :

```
halo {
dust
[ dust_type TIPO]
[ eccentricity eccentricità]
}
```

Man mano che i raggi passano attraverso la polvere, tutta la luce che arriva da qualunque sorgente

luminosa è accumulata e diffusa in relazione al tipo di polvere ed alla sua densità. Dal momento che questa accumulazione della luce include una prova per verificare l'occlusione della luce, altri oggetti possono proiettare ombre all'interno della polvere.
Con la polvere si possono usare gli stessi tipi di dispersione della luce che sono usati con l'atmosfera nel paragrafo "Atmosfera". Questi tipi sono :

```
#declare ISOTROPIC_SCATTERING = 1
#declare MIE_HAZY_SCATTERING = 2
#declare MIE_MURKY_SCATTERING = 3
#declare RAYLEIGH_SCATTERING = 4
#declare HENYEY_GREENSTEIN_SCATTERING = 5
```

La funzione di Henyey-Greenstein richiede il parametro aggiuntivo `eccentricity` che è descritto nel paragrafo riguardante l'atmosfera. Questa parola chiave può essere applicata solamente al tipo 5, la dispersione di Henyey-Greenstein.

7.6.4.3.3 Alone Emittente

Gli aloni emittenti emettono solamente luce. Ogni particella è una piccola sorgente luminosa. La luce che emette non è attenuata dalle altre particelle perché queste sono molto piccole.
Man mano che il raggio attraversa il campo di densità di una halo emittente, il colore delle particelle in ciascun elemento di volume e la loro variazione di trasparenza, vengono determinati mediante la mappatura di colore. Queste intensità sono accumulate per generare il colore totale dell'alone. L'intensità totale è aggiunta alla luce che passa attraverso l'alone. La luce dello sfondo è attenuata dalla densità totale dell'alone.
Dal momento che la luce emessa non è attenuata, può essere usata per modellare effetti come fuoco, esplosioni, raggi di luce, eccetera. Scegliendo una mappatura di colori appropriata, questi effetti possono essere resi con un alto grado di realismo.
Usando una mappatura planare si può raggiungere un alto grado di realismo per il fuoco. Per creare esplosioni è più opportuno invece usare una mappatura sferica e valori piuttosto alti di turbolenza (la cosa migliore è usare una mappa di colori periodica e frequenze maggiori di 1).
Gli aloni emittenti non proiettano alcuna luce sugli altri oggetti, comportandosi diversamente dalle sorgenti luminose, anche se sono composti da piccole particelle che emettono luce. Per farli realmente emettere luce, si dovrebbero usare centinaia o migliaia di piccole sorgenti luminose. Questo rallenterebbe il rendering in modo tale da renderlo inutilizzabile.

7.6.4.3.4 Alone Tralucente

L'alone tralucente è molto simile all'alone emittente. La differenza è che la luce emessa dalle particelle è attenuata dalle altre particelle. Si può pensarlo come una combinazione degli aloni attenuante e emittente.

7.6.4.4 Mappatura della Densità

La mappatura della densità è generalmente usata per mappare dei punti nello spazio a tre dimensioni su un intervallo ad una dimensione compreso tra 0.0 e 1.0, così da descrivere una distribuzione *spaziale* di particelle. I tipi differenti di mappature sono specificate da:

```
halo { planar_mapping | spherical_mapping |cylindrical_mapping |
box_mapping
...
}
```

Il tipo di mappatura predefinita è quella planare (`planar_mapping`). Poiché la mappatura è

riferita all'origine del sistema di coordinate, bisogna tenere presente questa regola: gli oggetti contenenti halo devono essere centrati sull'origine. Possono poi essere successivamente trasformati a seconda delle esigenze.
Ogni tipo di mappatura viene spiegato in dettaglio nei paragrafi seguenti.

7.6.4.4.1 Mappatura Cubica

La mappatura cubica (*Box Mapping*) può essere usata per creare una distribuzione di particelle a forma di parallelepipedo. La mappatura è calcolata prendendo il massimo dei valori assoluti dati per ogni coordinata secondo la formula:

```
r(x, y, z) = max(abs(x), abs(y), abs(z))
```

I valori maggiori di 1 verranno ridimensionati a 1.

7.6.4.4.2 Mappatura Cilindrica

La distanza r(x,y,z) dall'asse Y è data dalla formula:

```
r(x, y, z) = sqrt(x*x + z*z)
```

È usata per determinare i valori dell'intervallo. I valori più grandi di 1 verranno ridimensionati a 1.

7.6.4.4.3 Mappatura Planare

La distanza r(x,y,z) dal piano X-Z è data dalla formula:

```
r(x, y, z) = abs(y)
```

È usata per determinare i valori dell'intervallo. I valori più grandi di 1 verranno ridimensionati a 1.

7.6.4.4.4 Mappatura Sferica

La distanza r(x,y,z) dall'origine è data dalla formula:

```
r(x, y, z) = sqrt(x*x + y*y + z*z)
```

È usata per determinare i valori dell'intervallo. I valori maggiori di 1 verranno ridimensionati a 1.

7.6.4.5 Funzione Densità

La funzione densità (*Density Function*), determina come viene calcolata una mappatura di densità da un intervallo lineare (unidimensionale).
La funzione densità è specificata dalle seguenti parole chiave:

```
halo { [ constant | linear | cubic | poly ]
[ max_value MASSIMO]
[ exponent ESPONENTE]
}
```

La parola chiave `exponent` è usata solo insieme alla funzione densità `poly`.

Tutte le funzioni vengono descritte in questa sezione. In tutte le mappe il valore r(x,y,z) è calcolato in un campo compreso tra 0 e il massimo della densità (specificato dalla parola chiave `max_value`).

7.6.4.5.1 Costante

La funzione densità costante (`constant`) fornisce costantemente il valore `MAX_VALUE` nell'intervallo della mappa di densità. Questo viene calcolato dalla semplicissima formula `f(r) = MAX_VALUE`.

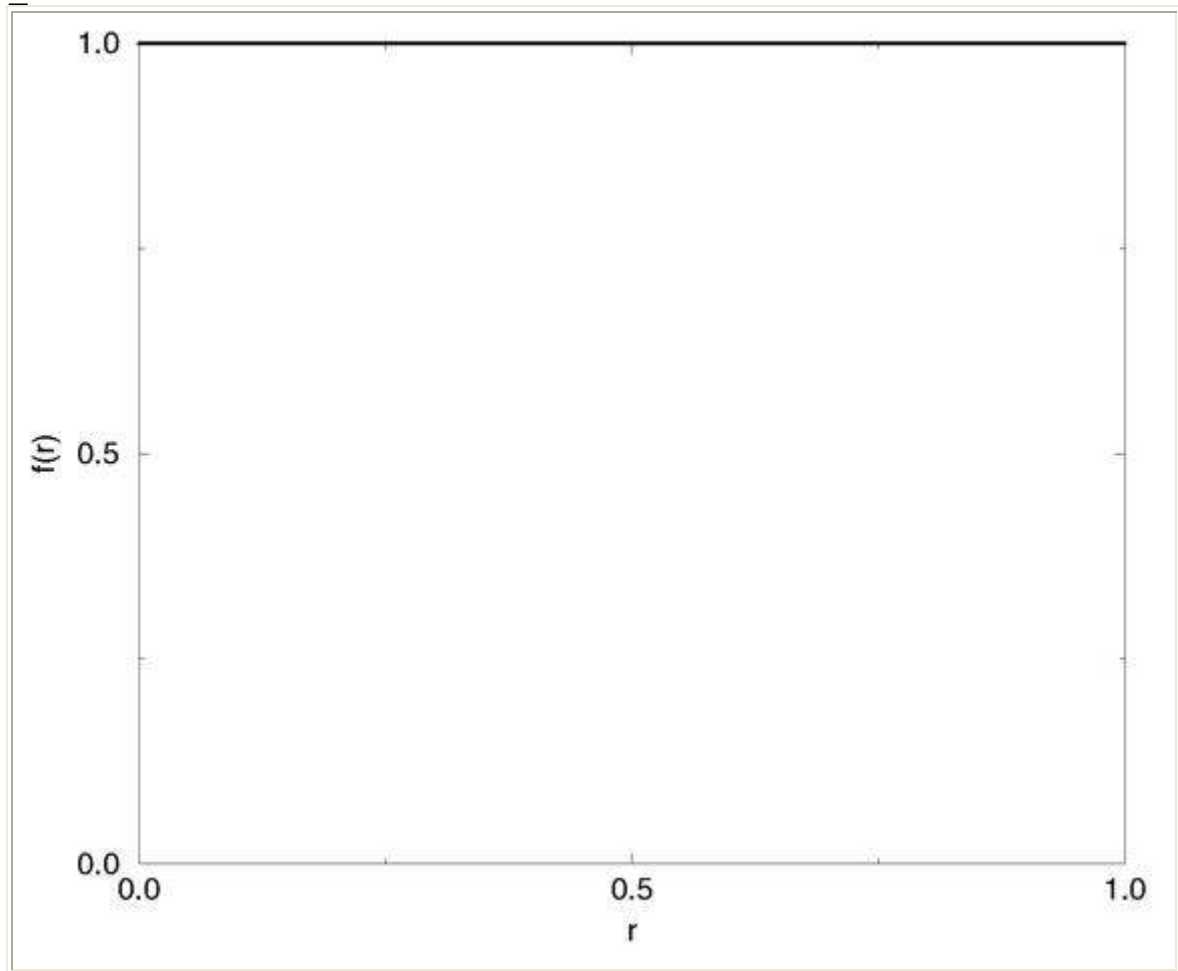



Figura 219- Funzione densità costante

La funzione di densità costante può essere usata per creare una distribuzione costante di particelle vincolate all'interno di un oggetto contenitore.

7.6.4.5.2 Lineare

Un passaggio lineare della densità dal valore `MAX_VALUE` in r=0 fino a 0 in r=1 si ottiene con la *funzione densità lineare*, indicata dalla parola chiave `linear`. Questa è data dalla formula:

```
f(r) = MAX_VALUE * (1 - r)
```

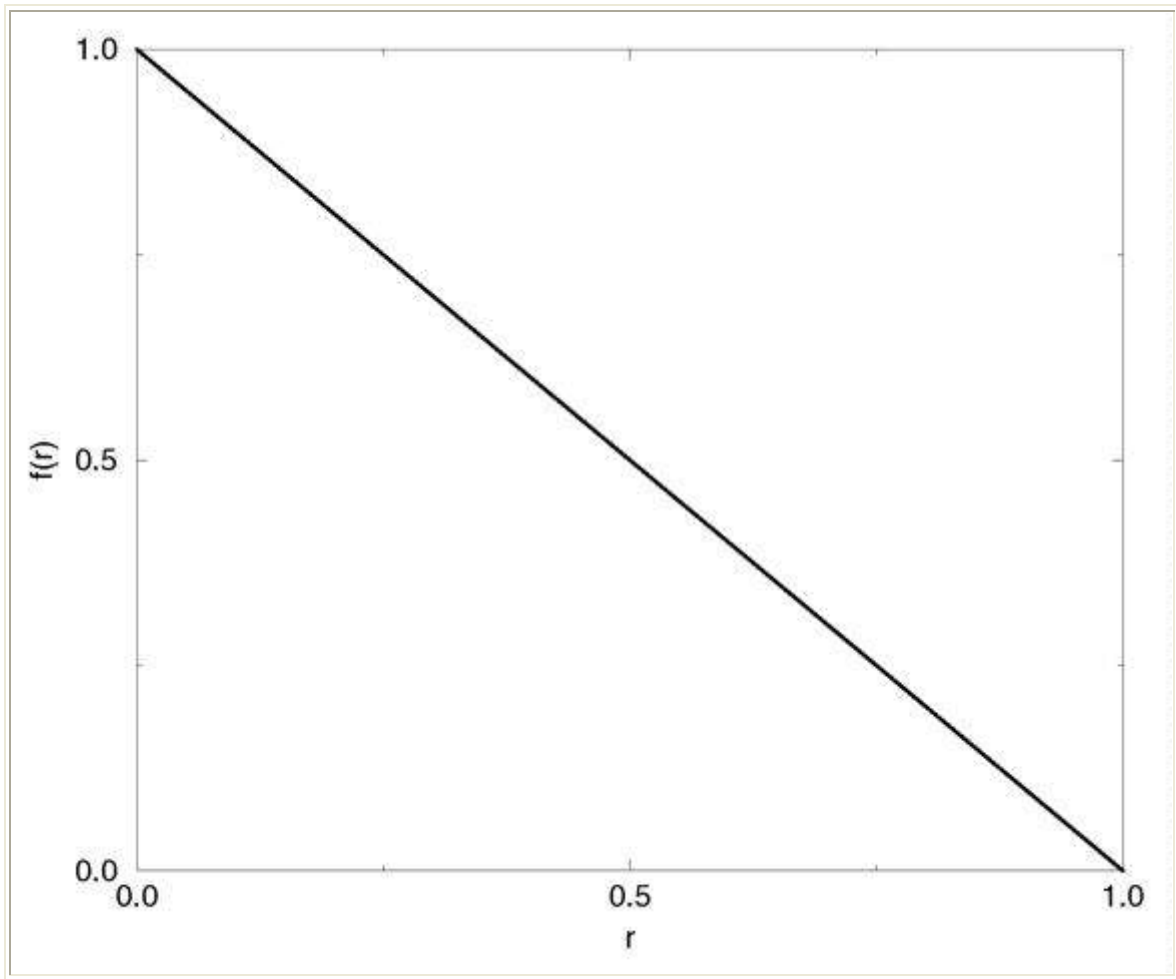



Fig. 220-Funzione densità lineare

7.6.4.5.3 Cubica

La funzione densità *cubica* rende più graduale il passaggio da `MAX_VALUE` in r=0 fino a 0 in r=1. . Questa è data dalla formula:

```
f(r) = MAX_VALUE * (2 * r - 3) * r * r + 1
```

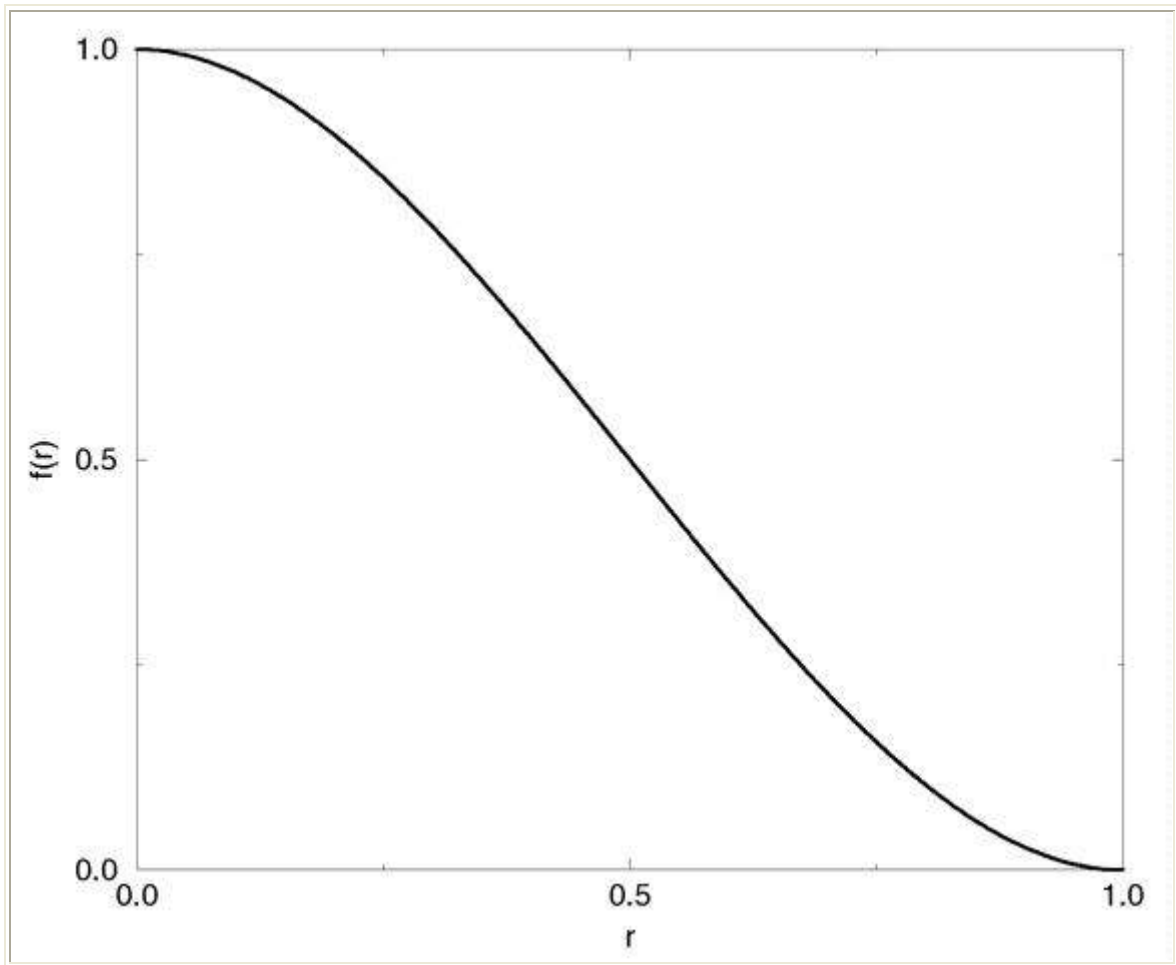



Figura 221-Funzione densità cubica

Questa curva è in effetti una spline cubica.

7.6.4.5.4 Polinomiale

Una funzione polinomiale può essere usata per creare una grande varietà di funzioni densità. Tutte hanno come massimo valore `MAX_VALUE` in r=0 e il minimo (cioè 0) in r=1. Questo è dato dalla formula:

```
f(r) = MAX_VALUE * (1 - r) ^ ESPONENTE
```

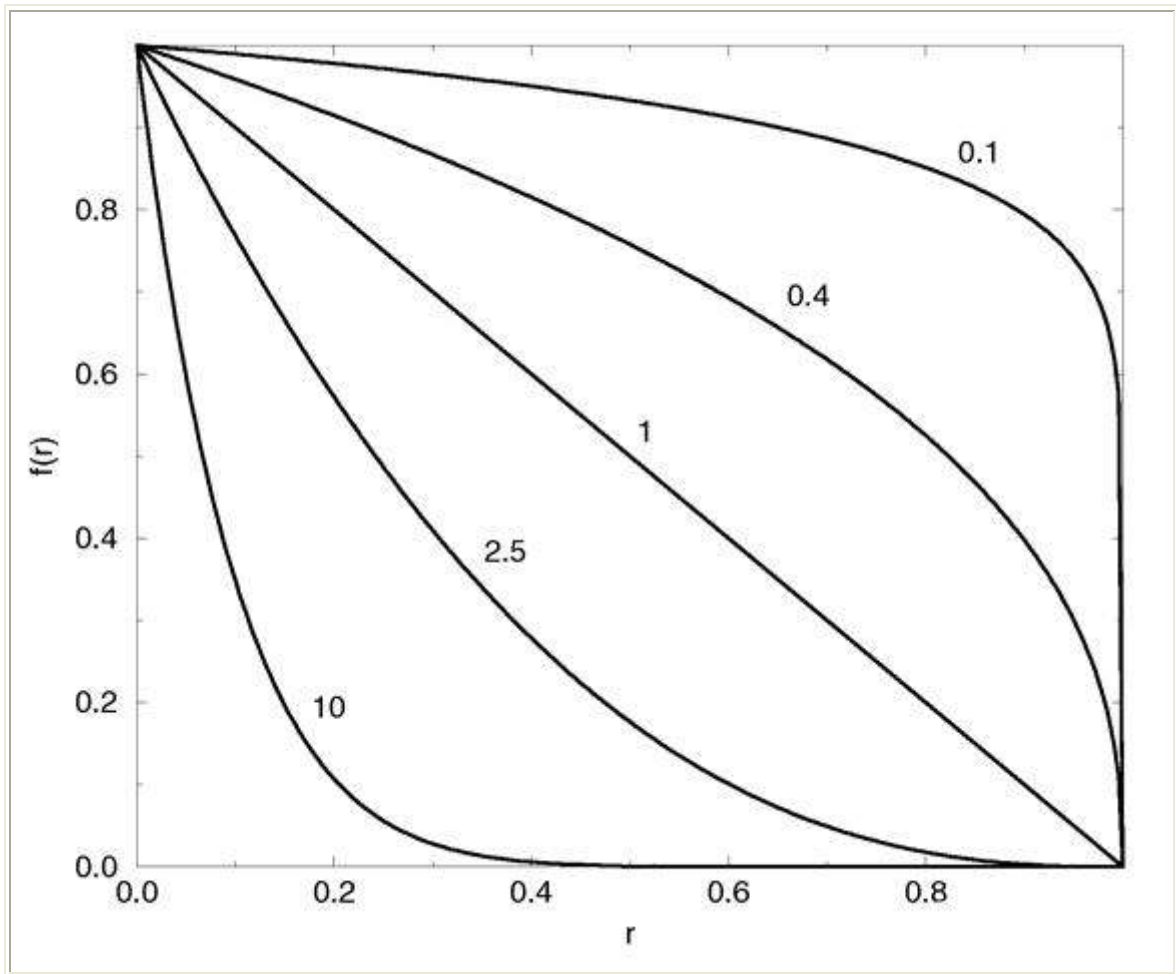



Fig. 222-Funzione polinomiale densità

L'esponente è determinato dalla parola chiave `exponent`. In caso di esponente uguale a 0 ci sarà una "caduta" lineare.

7.6.4.6 Mappa del Colore per gli Aloni

La densità `f(r)`, i cui valori variano tra 0 e `MAX_VALUE`, è 'mappata' sulla mappa dei colori (`color_map`) per ottenere i valori del colore e della trasparenza man mano che il raggio attraversa il campo di densità (inoltre il colore delle halo attenuanti viene calcolato dalla densità totale; leggere anche i paragrafi "Mappatura dell'Alone" e "Alone Attenuante"). La trasparenza della halo determinata per ogni valore di f(r) viene sommata o sottratta dalla trasparenza totale.

La mappa dei colori viene specificata da:

```
halo { [ colour_map MAPPA_DEI_COLORI ] }
```

La trasparenza viene registrata nel suo canale (T) per ogni elemento della mappa dei colori. Un semplice esempio è il seguente:

```
colour_map { [0 rgbt<1, 1, 1, 1>]
[1 rgbt<1, 1, 1, 0>] }
```

In questo esempio le aree con minore densità (piccolo f(r)) sono trasparenti, mentre quelle ad alta densità (grande f(r)) sono opache. Usando dei valori negativi di trasparenza, si possono creare dei

campi molto densi.

Nel caso della halo polvere, il canale della trasparenza è impiegato per specificare quanta luce *viene filtrata* dalla corrispondente voce della mappa dei colori. Per tutti gli altri tipi di halo questo valore viene ignorato.

Non esiste una mappa di colori predefinita.

7.6.4.7 Campionamento per gli Aloni

Gli aloni sono calcolati simulando un raggio di luce che attraversa un campo di densità. A fasi discrete, vengono presi campioni di colore dal campo di densità il loro valore viene valutato in accordo alla mappatura del colore ed a tutti gli altri parametri. Gli effetti dei singoli campioni sono quindi accumulati per calcolare l'effetto complessivo.
I seguenti parametri sono usati per regolare il processo di campionamento:

```
halo {
[ samples NUMERO DI CAMPIONI ]
[ aa_level LIVELLO DI ANTI-ALIASING ]
[ aa_threshold SOGLIA DI ANTIALIASING ]
[ jitter JITTER ]
}
```

Ogni singolo parametro viene descritto nelle seguenti sezioni.

7.6.4.7.1 Numero di Campioni

Il numero di campioni (*samples*) non è altro che il numero di raggi di luce da simulare. Un grande numero di campioni renderà l'halo più realistica allungando però il tempo di rendering.
Il valore di default di `samples` è 10. Questo valore è sufficiente per un semplice campo di densità che non usi effetti di turbolenza (`turbulence`).
Un'alta turbolenza o l'impiego di halo polvere (*dust halo*) necessitano di un grande numero di campioni per ottenere effetti realistici.

7.6.4.7.2 Sovracampionamento

Tutti i processi legati al campionamento, tendono per natura a dare risultati artificiosi, che si rivelano nell'immagine come delle scalettature, ad esempio, lungo le linee curve (*aliasing*). Un metodo per ridurre le artificiosità dovute all'aliasing è usare il sovracampionamento (*supersampling*), come avviene per il rendering dell'atmosfera**.** Se due raggi campione adiacenti riportano valori molto differenti tra loro, ne viene calcolato un terzo fra i due. Questo processo viene attivato solo se i raggi in questione differiscono almeno di un certo valore, detto di soglia (*threshold*), indicato dalla parola chiave `aa_threshold` e prosegue per un numero massimo di iterazioni `aa_level`, o fino a quando la differenza tra due pixel adiacenti non sia sotto il valore di soglia.
Di default il sovracampionamento non viene usato. I valori predefiniti di `aa_threshold` e `aa_level` sono rispettivamente 0.3 e 3.

7.6.4.7.3 Jitter

Il *jittering* può essere usato per deviare casualmente i campioni di una piccola quantità. Ciò può aiutare a ridurre le artificiosità dovute all'aliasing, al costo di aumentare il livello di rumore dell'immagine. Questo non è un problema grave dato che il nostro sistema visivo nota molto meno una dispersione casuale dell'errore, piuttosto che la sua distribuzione regolare. Di default la

funzione di jittering non è attiva. Il valore assegnato al parametro deve essere inferiore a 1.0.
E' da notare che il jittering si può usare anche se l'anti-aliasing non è utilizzato.

7.6.4.8 Modificatori degli Aloni

Questo paragrafo tratta tutti i modificatori delle halo di uso generale elencati sotto.

```
halo {
[ turbulence <TURBULENCE> ]
[ octaves OCTAVES ]
[ omega OMEGA ]
[ lambda LAMBDA ]
[ frequency FREQUENCY ]
[ phase PHASE ]
[ scale <VECTOR> ]
[ rotate <VECTOR> ]
[ translate <VECTOR> ]
}
```

7.6.4.8.1 Modificatore Frequency (frequenza)

Il parametro `frequency` definisce quante volte il campo di densità debba essere mappato su se stesso prima di venire mappato sulla mappa dei colori. La formula che calcola questa funzione è :

```
f_new(r) = (f(r) * FREQUENCY + PHASE) modulo 1.0
```

In questo modo la mappa dei colori verrà ripetuta tante volte quanto è il valore di `frequency`.

7.6.4.8.2 Modificatore Phase (Fase)

Il parametro `phase` determina di quanto deve essere sfasata la mappatura del campo di densità su se stesso. Vedi la formula del paragrafo precedente per l'uso di `phase`.
In questo modo il colore corrispondente a f(r)=0 può essere spostato a `phase*mod(1)`.

7.6.4.8.3 Modificatori di Trasformazione

Gli aloni possono essere trasformati usando le parole chiave `scale`, `rotate` e `translate`. Si deve fare attenzione a non spostare il campo di densità al di fuori dell'oggetto contenitore.

7.6.5 Texture Speciali

Le texture speciali sono texture complesse formate da più texture. Le texture componenti possono essere a loro volta texture semplici o speciali. Una texture semplice ha solo *una* frase `pigment{...}`, `normal{...}` e `finish{...}`ed eventualmente una definizione di halo. Anche le mappature di normali e pigmenti sono considerate texture semplici.
Le texture speciali usano la parola chiave `texture_map` per specificare l'unione di più texture oppure un'immagine, chiamata mappa dei materiali (`material_map`) con un metodo analogo alla mappatura di un'immagine.
Ci sono alcune limitazioni nell'uso delle texture speciali. Una texture speciale non può essere usata come texture predefinita (vedi il paragrafo "Istruzione Predefinita"), non può essere usata come strato in una texture stratificata, ma puoi usare texture stratificate come componenti di una texture speciale.

7.6.5.1 Mappe di Texture

Oltre a specificare colori che sfumano gli uni negli altri con le `color_map`, si possono sfumare anche le texture usando la parola chiave `texture_map`. La sintassi per una mappa di texture è identica a quella usata per la mappa di pigmenti, ma per ogni componente della mappa viene specificata una texture.
Una mappa di texture è specificata dalla seguente sintassi...

```
texture{
Tipo_di_pattern
texture_map {
[ NUM_1 TEXTURE_1]
[ NUM_2 TEXTURE_2]
[ NUM_3 TEXTURE_3]
...
}
modificatori...
}
```

Dove `NUM_1`, `NUM_2`... sono valori decimali compresi tra 0.0 ed 1.0 inclusi. Il termine `TEXTURE_1`, `TEXTURE_2`... indica tutto ciò che normalmente apparirebbe all'interno di una frase riguardante le texture, anche se non sono necessarie le parentesi graffe {...}, né la parola chiave `texture`. Le parentesi quadre, [...] *sono invece parte integrante* di questa istruzione, non indicano parti facoltative e circondano ogni elemento della mappa. Nella mappa ci possono essere da 2 a 256 elementi.
Ad esempio :

```
texture {
gradient x //questo è il tipo di pattern
texture_map {
[0.3 pigment{Red} finish{phong 1}]
[0.3 T_Wood11] //questo è un identificatore di texture
[0.6 T_Wood11]
[0.9 pigment{DMFWood4} finish{Shiny}]
}
}
```

Quando la funzione `gradient x` dà valori compresi tra 0.0 e 0.3, sarà usata la texture rossa. Da 0.3 a 0.6 sarà usata la texture `T_Wood11` ; da 0.6 a 0.9 ci sarà il passaggio graduale da `T_Wood11` a `DMFWood4`. Da 0.9 in poi, solo `DMFWood4`.
Le mappature di texture possono essere annidate fino a qualunque livello di complessità si voglia. Le texture che compaiono in una mappa possono avere mappature di colore o di texture , o qualunque tipo di texture si desideri.
Nell'area in cui due texture sfumano l'una nell'altra viene calcolato interamente il contributo di entrambe le texture e poi il colore risultante è calcolato con un'interpolazione lineare dei due. Ciò significa che la riflessione, la rifrazione e i calcoli relativi alla luminosità sono eseguiti due volte per ogni punto. Questo procedimento è diverso da ciò che accade durante il calcolo delle mappe di colore e normali nelle texture semplici, per le quali viene prima calcolato il colore, poi le normali ed infine, riflessione, rifrazione e luminosità sono calcolati una sola volta.
Intere texture possono essere usate con pattern a blocchi come `checker`, `hexagon`, `brick`. Per esempio :

```
texture {
```

```
checker
texture { T_Wood12 scale .8 }
texture {
pigment { White_Marble }
finish { Shiny }
scale .5
}
}
}
```

Per texture di questo tipo, si deve avere la frase `texture{...}` attorno alla definizione della texture stessa. E' anche da notare che questa sintassi proibisce l'utilizzo di texture stratificate, ma questo problema si può aggirare dichiarando un identificatore per la texture stratificata e usandolo all'interno della mappa. Una mappa di texture è usata anche per il pattern `average`. Vedi il paragrafo "Average (Media)" per dettagli.

7.6.5.2 Tiles

Le precedenti versioni di POV-Ray avevano una texture speciale chiamata `tiles` che creava un motivo di texture disposte a scacchiera. Anche se questa parola chiave è ancora valida per ragioni di compatibilità all'indietro, in questa versione si dovrebbe usare un blocco `checker{...}`, come descritto nel paragrafo "Mappe di Texture" al posto di `tiles`.

7.6.5.3 Mappe di Materiali

La texture speciale `material_map` estende il concetto della mappatura di un'immagine alle texture oltre che ai singoli colori. Una mappa dei materiali permette di avvolgere un motivo di texture a due dimensioni attorno agli oggetti a tre dimensioni. Invece di porre semplicemente il colore dell'immagine sull'oggetto viene specificata un'intera texture sulla base del numero d'ordine del colore nella tavolozza (*palette*) dell'immagine in quel punto. Si deve specificare una lista di texture da usare come una *tavolozza di texture* in sostituzione della normale tavolozza di colori dell'immagine. Quando si usano immagini in un formato che utilizza la tavolozza dei colori come GIF e alcuni tipi di immagini PNG e TGA il numero d'ordine del colore del punto è utilizzato per individuare una delle texture descritte. Per formati che non fanno uso della tavolozza dei colori si usa come indice il valore del componente rosso di ogni pixel. Se l'indice di un pixel è maggiore del numero delle texture presenti nella lista, allora viene rapportato a `modulo N`, dove `N` è la lunghezza della lista di texture.

7.6.5.3.1 Specificare una Mappa di Materiali

La sintassi di una mappa di materiali è :

```
texture {
material_map {
TIPO_FILE "nomefile"
MODIFICATORI_DELL'IMMAGINE...
texture {...} // Usata per il colore 0
texture {...} // Usata per il colore 1
texture {...} // Usata per il colore 2
texture {...} // Usata per il colore 3
// e così via.
}
TRASFORMAZIONI...
}
```

Dove `TIPO_FILE` è una delle seguenti parole chiave : `gif`, `tga`, `iff`, `ppm`, `pgm`, `png` o `sys`. Questa è seguita dal nome del file, definito mediante una qualsiasi espressione stringa valida. Alcuni modificatori facoltativi, descritti più avanti, possono seguire la specificazione del file. E' da notare che le prime versioni di POV-Ray permettevano che alcuni modificatori fossero inseriti *prima* dell'indicazione del tipo di file, ma questa sintassi è stata sorpassata da quella qui descritta. I file specificati nelle frasi `material_map{...}` saranno cercati nella directory corrente e, se non trovati, nelle directory specificate dal parametro `+L` o dall'opzione `Library_Path`. Questo rende preferibile tenere tutte le mappe in un'unica directory da specificare con `+L`. Attenzione : i percorsi di ricerca predefiniti del sistema operativo non vengono esaminati a meno che non siano specificati come 'library path' con il parametro `+L`.

Di default, il materiale è mappato sul piano x-y. Il materiale è proiettato sull'oggetto come se ci fosse un proiettore lungo l'asse z e copre interamente il quadrato che si estende fra le coordinate (0,0) e (1,1) senza tenere conto della dimensione originaria dell'immagine. Se vuoi cambiare quest'impostazione, puoi traslare, ruotare o scalare la texture per mapparla sulla superficie dell'oggetto come preferisci.
Il nome del file può essere seguito da uno o più modificatori di bitmap. Vedi il paragrafo "Modificatori di Immagini".
Dopo la frase `material_map`, ma sempre all'interno della frase `texture{...}`, è possibile applicare qualunque modificatore di texture. Non si possono aggiungere alla texture, al di fuori della mappa dei materiali, frasi di colore, normali, finitura o halo. La seguente sintassi non è permessa.

```
texture {
material_map {
gif "matmap.gif"
texture {T1}
texture {T2}
texture {T3}
}
finish {phong 1.0}
}
```

La finitura deve essere aggiunta ad ogni singola texture.
Le precedenti versioni di POV-Ray permettevano queste impostazioni, ma esse venivano ignorate. La suddetta restrizione sulla sintassi è stata resa necessaria per la correzione degli errori. Questo significa che alcune scene di POV-Ray 1.0, che usano mappe di materiali, possono aver bisogno di alcune modifiche che non possono essere apportate automaticamente con l'uso della parola chiave `version`.
Se alcuni particolari colori non sono usati in un'immagine, potrà essere necessario fornire al loro posto una "texture fasulla". In altre parole, potrà essere necessario usare un programma di disegno o altri programmi per esaminare la tavolozza dei colori dell'immagine e determinare come creare la lista delle texture.
Le texture all'interno di una mappa di materiali possono essere texture stratificate, ma texture che fanno uso di una mappa di materiali non possono diventare strati di altre texture stratificate. Per usare una texture stratificata all'interno di una mappa di materiali è necessario dichiararla con un identificatore di texture e richiamarla all'interno della lista delle texture.

7.6.6 Texture Stratificate

E' possibile creare diversi effetti speciali usando texture stratificate. Una texture stratificata è costituita da diverse texture che sono parzialmente trasparenti, disposte una sopra l'altra per creare texture più complesse. I diversi strati di texture sono visibili attraverso le parti trasparenti e contribuiscono a creare l'aspetto di un'unica texture che è in realtà la combinazione di diverse altre. Si possono creare texture stratificate inserendo in una lista due o più texture una dopo l'altra. L'ultima texture nella lista costituirà lo strato superiore e la prima lo strato inferiore. Tutte le texture in una texture stratificata, tranne lo strato più basso, dovranno essere almeno in parte trasparenti. Per esempio :

```
object {
My_Object
texture {T1} // strato inferiore
texture {T2} // strato semitrasparente
texture {T3} // strato superiore semitrasparente
}
```

Nell'esempio, T2 sarà visibile solo dove T3 è trasparente e T1 sarà visibile solo dove T2 e T3 sono trasparenti.
Il colore degli strati sottostanti è filtrato dagli strati superiori, ma il risultato non apparirà come una serie di superfici trasparenti. Se avessi una pila di superfici con le texture applicate a ciascuna di esse, la luce sarebbe filtrata due volte : una volta all'entrata, quando gli strati inferiori sono illuminati dalla luce filtrata ed una all'uscita. Le texture stratificate non filtrano la luce in entrata. Altre parti dei calcoli dell'illuminazione funzionano in modo diverso. Il risultato è ottimo e permette di ottenere bellissime texture, ma queste sono diverse da vere e proprie superfici sovrapposte. Vedi il file **stones.inc** nella directory dei file INC per alcune splendide texture stratificate.
E' da notare che le texture stratificate devono fare uso della parola chiave `texture{...}` attorno ad ogni frase di pigmento, normale o finitura. Non si possono usare più frasi di pigmento, normale o finitura senza includerle nella frase `texture{...}`.
Le texture stratificate possono essere dichiarate. Per esempio :

```
#declare Esempio_Strat =
texture {T1}
texture {T2}
texture {T3}
```

può essere richiamato come segue :

```
object {
Coso
texture {
Esempio_Strat
// tutti i pigmenti, normali o finiture qui
// cambiano solo lo strato inferiore
}
}
```

Se vuoi usare una texture stratificata in un pattern modulare, come `checker`, `hexagon` o `brick`, o in una mappa dei materiali, devi prima dichiarare la texture e poi richiamarla all'interno di una singola frase di texture. Una texture speciale non può essere usata come strato in una texture stratificata. Comunque puoi usare texture stratificate come un qualunque componente di una texture speciale.

7.6.7 Motivi (Pattern)

POV-Ray usa un metodo detto "*Three dimensional solid texturing"* per definire il colore, la rugosità e le altre proprietà di una superficie. Il modo in cui una texture varia su una superficie è specificato dal motivo (*pattern*) usato. I motivi sono usati all'interno delle frasi dei pigmenti, delle normali e delle mappe di texture. Tutti i motivi in POV-Ray sono tridimensionali : per ciascun punto dello spazio ciascun motivo ha uno specifico valore. I motivi non circondano le superfici come se venisse avvolta un'immagine attorno agli oggetti ; esistono nello spazio tridimensionale e gli oggetti sono intagliati in essi, come un oggetto può essere intagliato in un blocco di legno o di pietra.
Consideriamo un pezzo di legno. Contiene bande chiare e scure costituite da cilindri concentrici (gli anelli di crescita). Osservando un tronco di legno alle estremità vediamo dei cerchi concentrici, mentre lungo la sua altezza vediamo delle linee che costituiscono le venature. Comunque, il motivo è presente nell'intero blocco. Se tagli o scolpisci il legno, metterai alla luce lo stesso tipo di motivo, che infatti è anche all'interno ; proprio come una cipolla, che è costituita da sfere concentriche che sono visibili solo quando viene tagliata. Il marmo è costituito da strati ondulati di sedimenti colorati che si sono trasformati in roccia. Questi motivi solidi possono essere simulati usando funzioni matematiche. Anche altri motivi casuali come `granite` o `bumps` o `dents` possono essere generati usando un algoritmo che calcola valori casuali.
In ogni caso, le coordinate x, y, z di un punto della superficie sono utilizzate per calcolare delle funzioni matematiche che danno come risultato un valore decimale. Quando sono usati con mappe di colori o di pigmenti, quel valore assumerà il colore del pigmento da usare. Nelle frasi di descrizione delle normali, la funzione corrispondente al pattern usato, modificherà il vettore normale alla superficie per dare a quest'ultima un'apparenza irregolare. Usati con mappe di texture, le funzioni determineranno quale combinazione delle texture deve essere usata.
I paragrafi seguenti descrivono ognuno di questi pattern. Vedi i paragrafi "Pigmento" e "Normali" per maggiori dettagli sull'uso dei pattern.

7.6.7.1 Agate (Agata)

Il motivo `agate` è un motivo simile al marmo (`marble`) ma che usa una particolare funzione, diversa da quella tradizionale, per aggiungere la turbolenza. Anche la turbolenza tradizionale può essere usata, ma normalmente non è necessaria perché `agate` è già molto irregolare. Puoi controllare la quantità di turbolenza aggiungendo la parola chiave `agate_turb` seguita da un numero decimale. Per esempio :

```
pigment {
agate
agate_turb 0.5
color_map {
...
}
}
```

Il pattern `agate` usa una forma d'onda a dente di sega (*ramp_wave*) per default, ma può utilizzare qualunque tipo d'onda. Questo motivo può essere usato con mappe di colore, di pigmenti, di normali, mappe slope e mappe di texture.

7.6.7.2 Average (Media)

Tecnicamente `average` non è un tipo di pattern vero e proprio, ma è elencato qui perché la sintassi è simile a quella degli altri pattern. Normalmente un pattern specifica come i colori o le normali sono scelti da una mappa, mentre `average` fa calcolare a POV-Ray la media di tutti pattern che vengono specificati. `Average` era stato originalmente ideato per l'uso in una frase di

descrizione delle normali insieme ad una mappa di normali, come metodo per usare più di un pattern di normali sulla stessa superficie. Comunque `average` può essere usato in una frase `pigment{...}` insieme ad una mappa di pigmenti o in una frase `texture{...}` insieme ad una mappa di texture. Quando è usato con i pigmenti, la sintassi è :

```
pigment {
average
pigment_map
{
[PESO_1 PIGMENTO_1]
[PESO_2 PIGMENTO_2]
...
[PESO_n PIGMENTO_n]
}
MODIFICATORI
}
```

Allo stesso modo si può usare una mappa di texture in una frase `texture{...}`. Tutte le texture sono calcolate per intero, i colori vengono pesati e ne viene fatta la media. Quando viene usata con una mappa di normali, vengono calcolate più copie della superficie originaria. Queste sono perturbate da ogni pattern di normali presente nella lista ed infine vengono pesate, sommate e normalizzate. Vedi i paragrafi "Mappe di pigmento", "Mappe normali" e "Mappe di texture".

7.6.7.3 Bozo

Il pattern `bozo` è una funzione casuale che, con l'aggiunta di una leggera turbolenza, è di solito usata per creare nuvole. Il pattern `spotted` è identico a `bozo`, ma nelle prime versioni di POV-Ray non gli si poteva aggiungere turbolenza. Ora la turbolenza può essere aggiunta a qualunque pattern quindi `spotted` e `bozo` sono uguali in tutto, ma si è mantenuto `spotted` per ragioni di compatibilità all'indietro. Anche il pattern `bumps` è uguale a `bozo` tranne quando è usato in una frase `normal{...}`. In questo caso `bumps` usa un metodo leggermente diverso per perturbare il vettore normale alla superficie. La funzione `bozo` ha le seguenti proprietà :
1) è definita nello spazio a tre dimensioni, cioè legge i valori di x, y e z e dà il valore del rumore in quel punto.
2) se due punti sono lontani, i loro valori del rumore sono relativamente casuali
3) se due punti sono vicini, i loro valori di rumore sono simili.
Si può visualizzare questo pattern immaginando una stanza ed un termometro la cui scala si estende tra 0 e 1. Ogni punto della stanza ha una temperatura. I punti lontani fra loro hanno temperature casuali, mentre i punti vicini hanno temperature vicine. La temperatura cambia gradualmente ma casualmente in tutto il volume della stanza. Ora mettiamo nella stanza un oggetto e un pittore. Il pittore misura la temperatura su ogni punto dell'oggetto e dipinge quel punto di un colore diverso a seconda della temperatura. Cosa otteniamo ? Un oggetto con una texture bozo !
Il pattern `bozo` usa un'onda `ramp_wave` per default, ma può usare qualunque tipo di onda. Questo pattern può essere usato con mappe di colore, pigmento, normali, texture e `slope_map`.

7.6.7.4 Brick (Mattoni)

Il pattern `brick` genera una superficie ricoperta di mattoni rettangolari. I mattoni sono sfalsati di una lunghezza pari a mezzo mattone ogni riga nella direzione x-z. Uno strato di "malta" circonda ogni mattone. La sintassi è :

```
pigment {
```

```
brick COLORE_1, COLORE_2
brick_size VETTORE
mortar DECIMALE //MALTA
}
```

dove COLORE_1 è il colore della malta e COLORE_2 è il colore dei mattoni. Se non vengono specificati colori, il programma usa un rosso scuro e un grigio scuro predefiniti.
La dimensione di default di un mattone con la malta è <8, 3, 4.5>. Lo spessore predefinito della malta è di 0.5 unità. Questi valori possono essere cambiati usando i modificatori di pattern brick_size e mortar. Puoi anche usare frasi di pigmento al posto dei colori. Per esempio:

```
pigment {
brick pigment{Jade}, pigment{Black_Marble}
}
```

Quando viene usato con le normali, la sintassi è :

```
normal {
brick QUANTITA'
}
```

Dove QUANTITA' è un valore decimale facoltativo per definire la dimensione delle asperità della superficie. Si possono usare anche intere frasi di normali con brick. Per esempio:

```
normal {
brick normal{bumps 0.2}, normal{granite 0.3}
}
```

Quando è usato con texture, la sintassi è :

```
texture {
brick texture{T_Gold_1A}, texture{Stone12}
}
```

Il pattern brick non può usare i modificatori delle forme d'onda, mappe di colori o slope_map.

7.6.7.5 Bumps

Il pattern bumps è stato creato originariamente per essere usato come un pattern delle normali. Usa una funzione casuale che crea vasti rigonfiamenti (*bumps*) della superficie quando è ingrandita e ha l'aspetto di una buccia d'arancia quando è rimpicciolita. Di solito i *bumps* sono distanti circa un'unità. Quando è usato come una normale, bumps usa una funzione di perturbazione particolare. Questo significa che questo pattern non può essere usato con le mappe di normali, le slope_map e le varie forme d'onda. Quando è usato come un pattern di pigmenti o di texture, è identico a bozo o a spotted e simile ai bumps delle normali, ma non identico, come è invece la maggior parte dei pattern delle normali quando vengono applicati ad un pigmento. Quando è usato in frasi di pigmento o texture, il pattern bumps usa l'onda ramp_wave per default, ma può usare qualunque tipo di onda. Il pattern bumps può essere usato con mappe di colore, pigmento, normali e texture.

7.6.7.6 Checker (Scacchiera)

Il pattern checker produce un motivo a scacchiera consistente di quadrati alternati dei colori COLORE_1 e COLORE_2. Se non sono specificati colori, la scacchiera sarà blu e verde. La sintassi è :

```
pigment { checker COLORE_1, COLORE_2 }
```

Il pattern checker è in effetti una serie infinita di cubi di lato 1 disposti nello spazio in modo che siano alternati in ogni direzione. Se si assegna questo pattern ad un qualunque oggetto, lo si vedrà come se fosse stato intagliato in questa infinità di cubi.
Si possono anche usare frasi di descrizione dei pigmenti al posto dei semplici colori. Ad esempio:

```
pigment { checker pigment{Jade}, pigment{Black_Marble} }
```

Quando viene usato con le normali, la sintassi è :

```
normal {checker QUANTITA'}
```

dove QUANTITA' e un valore facoltativo che specifica la profondità dei bumps. All'interno di checker si possono usare anche intere descrizioni delle normali. Ad esempio :

```
normal {
checker normal{gradient x scale .2},
normal{gradient y scale .2}
}
```

Quando si usa con le texture, la sintassi è invece :

```
texture { checker texture{T_Wood_3A},texture{Stone12} }
```

Questo utilizzo di checker nelle texture sostituisce la parola chiave tiles delle versioni precedenti di POV-Ray. Si può ancora usate tiles ma potrebbe venire eliminata nelle prossime versioni di POV-Ray, quindi la scelta migliore è utilizzare checker.
Il checker è un pattern che non può usare i modificatori delle forme d'onda, mappe di colori o slope_map.

7.6.7.7 Crackle (Spaccature)

Il pattern crackle è un insieme di poligoni disposti casualmente che suddividono lo spazio. Ingrandito e senza turbolenza, rende bene un pavimento o un muro di pietra. Rimpicciolito e senza turbolenza, rende bene l'aspetto di una ceramica leggermente crepata. Usare alti valori della turbolenza lo rende un buon marmo che evita anche il problema della presenza di strati apparentemente uguali nel pigmento marble.
Matematicamente, l'insieme crackle(p)=0 è il *diagramma di Voronoi* di un campo di punti semi casuali e crackle(p)<0 è la distanza dall'insieme lungo il percorso più breve (un diagramma di Voronoi è il luogo dei punti equidistanti dai due punti a loro più vicini in un insieme di punti separati, come la pellicola della schiuma di sapone rispetto al centro delle bolle.
Il pattern crackle usa l'onda ramp_wave di default, ma può usare qualunque tipo di onda. Il pattern crackle può essere usato con mappe di colore, pigmento, normali, texture e slope_map.

7.6.7.8 Dents

Il pattern `dents` era progettato in origine per essere utilizzato con le normali. E' particolarmente interessante se viene usato con texture metalliche, poiché dà l'impressione che sulla superficie siano state prodotte delle scalfitture con un martello. Normalmente, queste scalfitture sono distanti circa un'unità. Quando viene usato per le normali, `dents` usa una funzione di perturbazione *specializzata*. Ciò significa che non può essere usato con modificatori di mappe di normali, `slope_map` o di forma d'onda in una frase `normal{...}`.
Quando viene usato come un pattern da applicare a pigmenti o texture, `dents` è simile alla versione usata per le normali, ma non identico. Quando viene usato per colori, pigmenti e texture, il pattern `dents` usa l'onda `ramp_wave`, ma può usare qualunque tipo di onda. Il pattern `dents` può essere usato con mappe di colore, pigmento, normali e texture.

7.6.7.9 Gradient (Gradiente)

Uno dei pattern più semplici è il pattern a gradiente, specificato dalla parola chiave `gradient` nel modo seguente :

```
pigment {gradient VETTORE}
```

dove `VETTORE` è un vettore rivolto nella direzione in cui sfuma il colore. Ad esempio,

```
pigment { gradient x } // le fasce di colore variano
// lungo l'asse x
```

produce una serie di fasce colorate che sembrano strati di colori diversi accostati gli uni agli altri. I punti ad x=0 hanno il primo colore della mappa, spostandosi verso x=1 il colore sfuma nell'ultimo colore della mappa, per poi riprendere dal primo ad x=1 e raggiungere di nuovo il secondo ad x=2. Il pattern si rovescia per valori negativi di x. Usare `gradient y` o `gradient z` fa sfumare i colori lungo gli assi y e z rispettivamente. Si può usare qualunque vettore, ma x, y e z sono i più comuni. Quando viene usato nelle normali, `gradient` genera un motivo simile ad un'onda a dente di sega. La sintassi è :

```
normal { gradient VETTORE, QUANTITA'}
```

dove il vettore che specifica l'orientazione del motivo è necessario, mentre il numero decimale che specifica l'ammontare dell'effetto è facoltativo. Il pattern `gradient` usa l'onda `ramp_wave`, ma può usare qualunque tipo di onda. Il pattern `gradient` può essere usato con mappe di colore, pigmento, normali, texture e slope.

7.6.7.10 Granite (Granito)

Questo pattern fa uso di una semplice funzione di rumore frattale di tipo 1/x per generare una superficie simile al granito. Questo pattern è utilizzato con alcune interessanti mappe di colori nel file **stones.inc** per creare eccezionali texture stratificate che riproducono pietre di vario tipo.
Se usato come un pattern di normali crea una superficie estremamente irregolare che assomiglia ad una strada sterrata, o a roccia grezza.
Il pattern `granite` usa l'onda `ramp_wave` di default, ma può usare qualunque tipo di onda. Il pattern `granite` può essere usato con mappe di colore, pigmento, normali, texture e slope.

7.6.7.11 Hexagon (Esagoni)

Il pattern `hexagon` è un pattern a blocchi come `checker` e `brick` che genera un motivo ad esagoni affiancati disposti lungo il piano x-y. Immagina delle bacchette di sezione esagonale

disposte parallelamente e raggruppate come nella figura sotto. Per questo pattern si devono specificare tre diversi colori :

```
pigment { hexagon COLORE_1, COLORE_2, COLORE_3 }
```

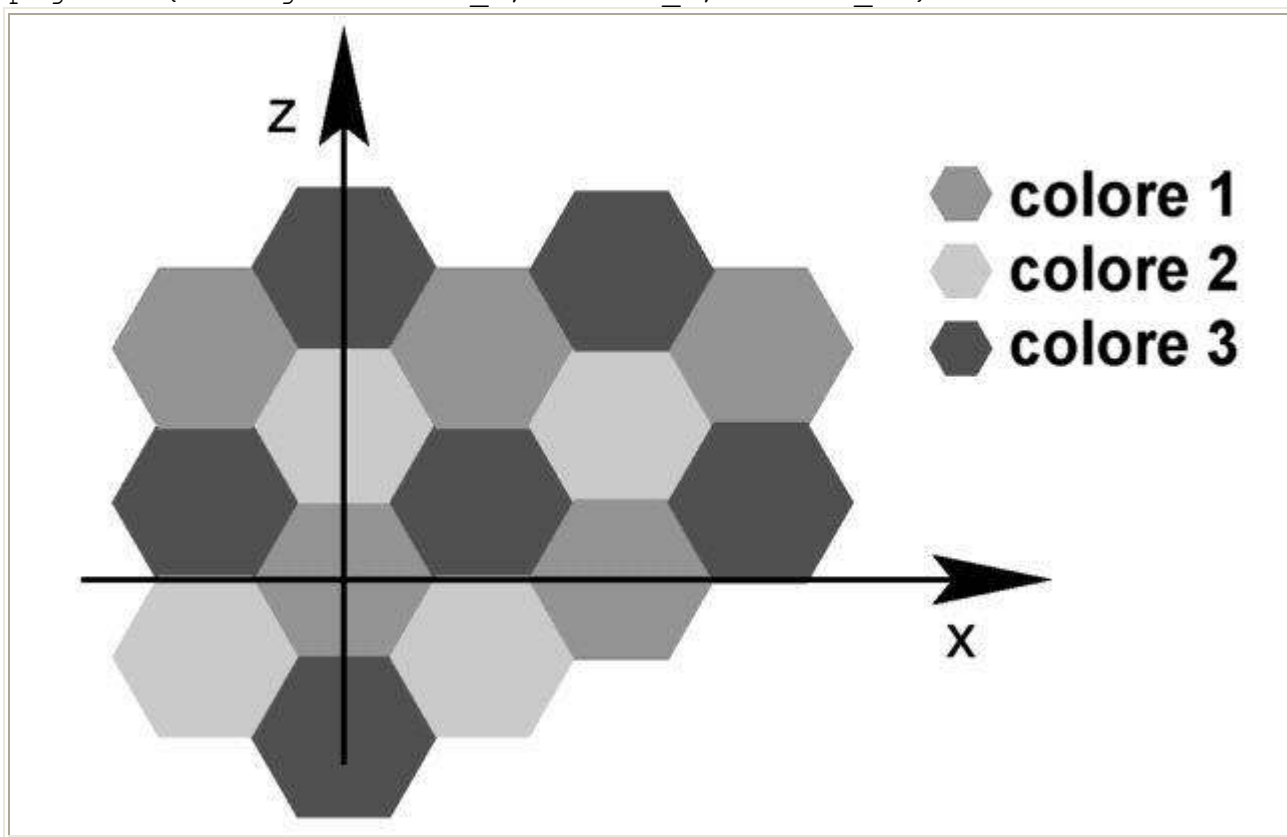



Fig. 223- Il pattern hexagon

I tre colori si ripeteranno nel pattern con l'esagono del colore 1 centrato sull'origine, il secondo spostato nella direzione +z ed il terzo verso -z. Ogni lato degli esagoni è lungo 1. Le 'bacchette' si estendono infinitamente lungo la direzione parallela all'asse y. Se non vengono specificati i colori, POV-Ray usa l'impostazione di default : un piano ad esagoni blu, verdi e rossi.
Si possono usare anche frasi `pigment{...}` al posto dei singoli colori, come ad esempio :

```
pigment {
hexagon pigment { Jade },
pigment { White_Marble },
pigment { Black_Marble }
}
```

Quando viene usato con normali, la sintassi è la seguente :

```
normal { hexagon QUANTITA' }
```

dove `QUANTITA'` è un numero decimale facoltativo che indica la dimensione dei rilievi. Si possono utilizzare anche intere descrizioni di normali, come

```
normal {
hexagon
normal { gradient x scale .2 },
normal { gradient y scale .2 },
```

```
normal { bumps scale .2 }
}
```

Quando è usato con texture, la sintassi è invece

```
texture {
hexagon
texture { T_Gold_3A },
texture { T_Wood_3A },
texture { Stone12 }
}
```

Questo pattern non può usare forme d'onda, mappe di colori o mappe slope.

7.6.7.12 Leopard (Leopardo)

Il pattern `leopard` crea un motivo geometrico di macchie circolari. Il pattern `leopard` usa l'onda `ramp_wave` di default, ma può usare qualunque tipo di onda. Il pattern `leopard` può essere usato con mappe di colore, pigmento, normali, texture e slope.

7.6.7.13 Mandel

Il pattern `mandel` calcola il classico insieme di Mandelbrot e lo proietta sul piano x-y. Per calcolare l'insieme di Mandebrot usa le coordinate x ed y. Il pattern `mandel` si indica con la parola chiave `mandel` seguita da un numero intero che definisce il numero massimo di iterazioni da usare per il calcolo. Valori tipici vanno da 10 a 256, ma si può usare qualsiasi numero intero. Ad esempio :

```
pigment {
mandel 25
color_map {
[0.0 color Cyan]
[0.3 color Yellow]
[0.6 color Magenta]
[1.0 color Cyan]
}
scale .5
}
```

Il valore passato alla mappa dei colori è dato dalla formula

```
valore=num_iterazioni/iterazioni_massime
```

Quando lo si usa come un pattern di normali, la sintassi è :

```
normal {
mandel ITER, QUANTITA'
}
```

dove il numero intero `ITER` è necessario ed il numero decimale `QUANTITA'`, facoltativo, rappresenta la dimensione apparente dei rilievi.
Il pattern si estende infinitamente nella direzione z in modo simile alla mappatura planare di un'immagine. Il pattern `mandel` usa l'onda `ramp_wave` di default, ma può usare qualunque tipo

di onda. Il pattern `mandel` può essere usato con mappe di colore, pigmento, normali, texture e slope.

7.6.7.14 Marble (Marmo)

Il pattern `marble` è molto simile a `gradient x`. Il pattern `gradient x` usa però un'onda di tipo `ramp_wave` (a dente di sega), che significa che usa il colore all'indice 0.0 della mappa dei colori per x=0, poi sfuma gradualmente al colore di indice 1.0 per x=1, quindi 'salta' nuovamente al colore di indice 0.0 e ripete il motivo all'infinito. Il pattern `marble` usa invece un'onda di forma triangolare, (`triangle_wave`) in cui usa la mappa dei colori da 0 ad 1 tra x=0 ed x=0.5, poi la 'rovescia' e torna indietro da x=0.5 ad x=1. Ad esempio :

```
pigment {
gradient x
color_map {
[0.0 color Yellow]
[1.0 color Red]
}
}
```

sfuma da giallo a rosso e quindi torna bruscamente al giallo. Sostituendo `gradient` con `marble` si sfuma dal giallo al rosso tra 0 e 0.5 e di nuovo dal rosso al giallo tra 0.5 ed 1. Le prime versioni di POV-Ray non permettevano di modificare il tipo di onda. Ora che è possibile modificarlo su quasi tutti i tipi di pattern, le differenze tra `gradient` e `marble` si limitano solo alla forma d'onda predefinita.
Utilizzato con la turbolenza ed una mappa dei colori appropriata, questo pattern simula le venature di marmo, giada o di altre pietre. Il pattern `marble` può essere usato con mappe di colore, pigmento, normali, texture e slope.

7.6.7.15 Onion (Cipolla)

Il pattern `onion` è un motivo di sfere concentriche, disposte come gli strati di una cipolla. Ogni strato ha lo spessore di un'unità. Il pattern `onion` usa l'onda `ramp_wave` di default, ma può usare qualunque tipo di onda. Il pattern `onion` può essere usato con mappe di colore, pigmento, normali, texture e slope.

7.6.7.16 Quilted (Trapunto)

Il pattern `quilted` era stato originariamente progettato come pattern per le normali. Ha questo nome poiché può creare un motivo in qualche modo rassomigliante alla superficie di una coperta trapunta, o ad un pavimento. I quadrati che lo formano sono in effetti cubi a tre dimensioni di lato unitario.
Quando viene usato per il calcolo delle normali, utilizza una funzione *specializzata*, il che significa che `quilted` non può essere utilizzato con modificatori di normali, mappe slope o modificatori del tipo d'onda. Quando usato come pattern di pigmenti o di texture, `quilted` è simile a quello che si usa per le normali, ma non identico come sono invece molti dei pattern di normali quando applicati ai pigmenti. Quando è usato all'interno di una frase `pigment{...}` o `texture{...}`, il pattern `quilted` usa la forma d'onda `ramp_wave`, ma può usare qualunque tipo di onda. Questo pattern può essere usato insieme a mappe di colore, pigmenti e texture.
I due parametri `control0` e `control1` hanno la funzione di controllare la curvatura delle 'trapungiture'. La sintassi è :

```
normal {
```

```
quilted QUANTITA'
control0 C0
control1 C1
}
```

I valori dovrebbero in generale essere mantenuti tra 0.0 ed 1.0. Il valore di default è 1.0. In sezione, queste 'trapungiture' sono così :

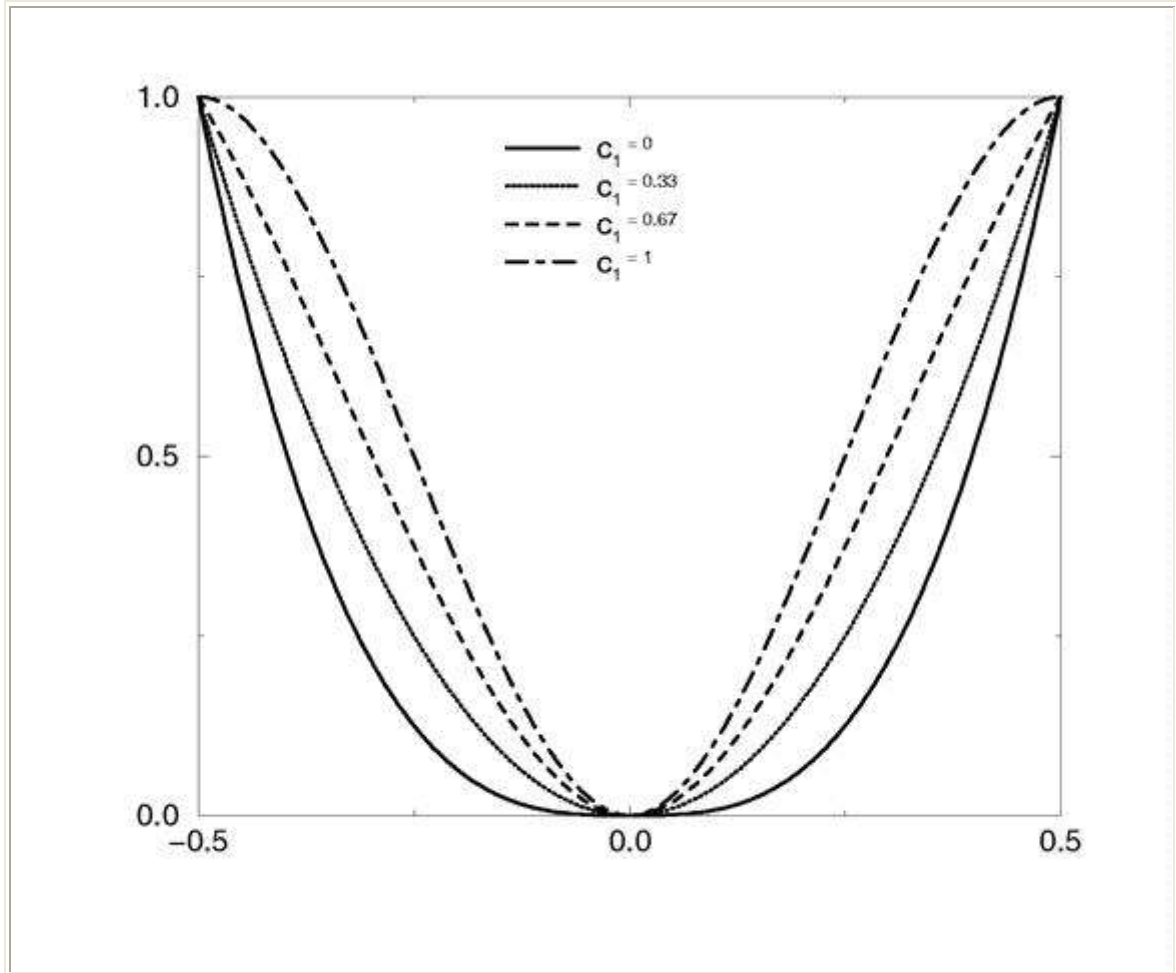



Fig. 224-Il pattern 'quilted' con c0=0 e diversi valori di c1.

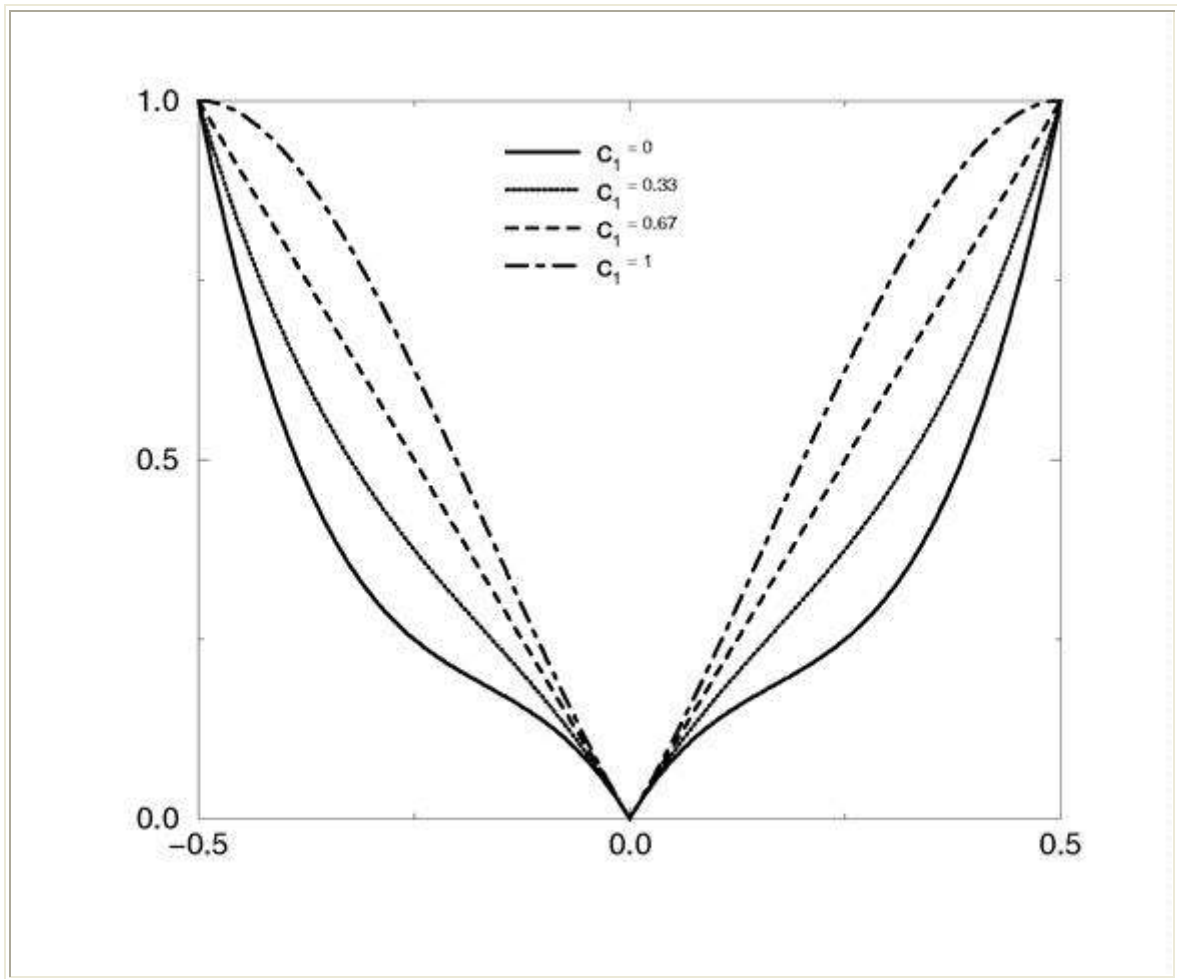



Fig.225 -Il pattern 'quilted' con c0=0.33 e diversi valori di c1.

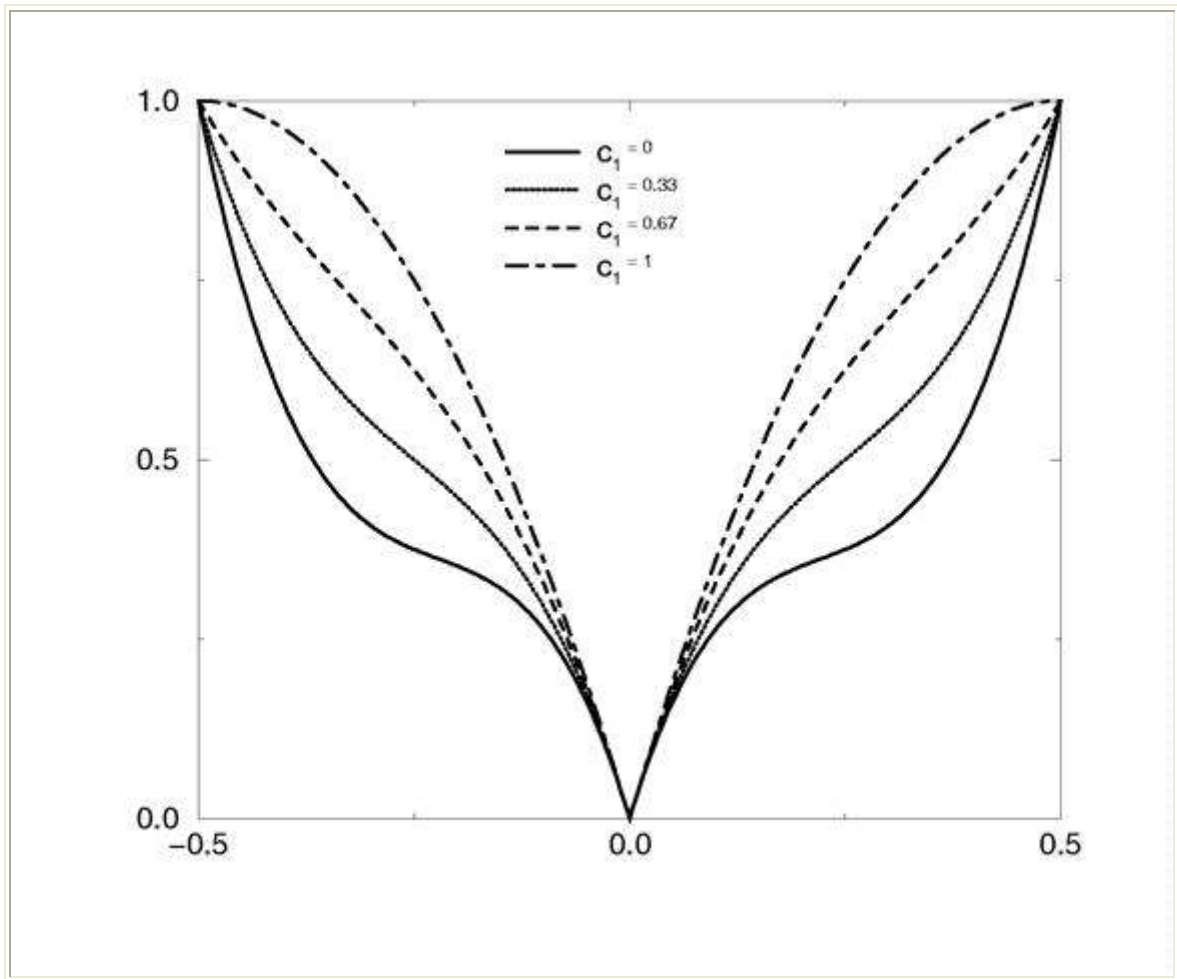



Fig. 226-Il pattern 'quilted' con c0=0.67 e diversi valori di c1.

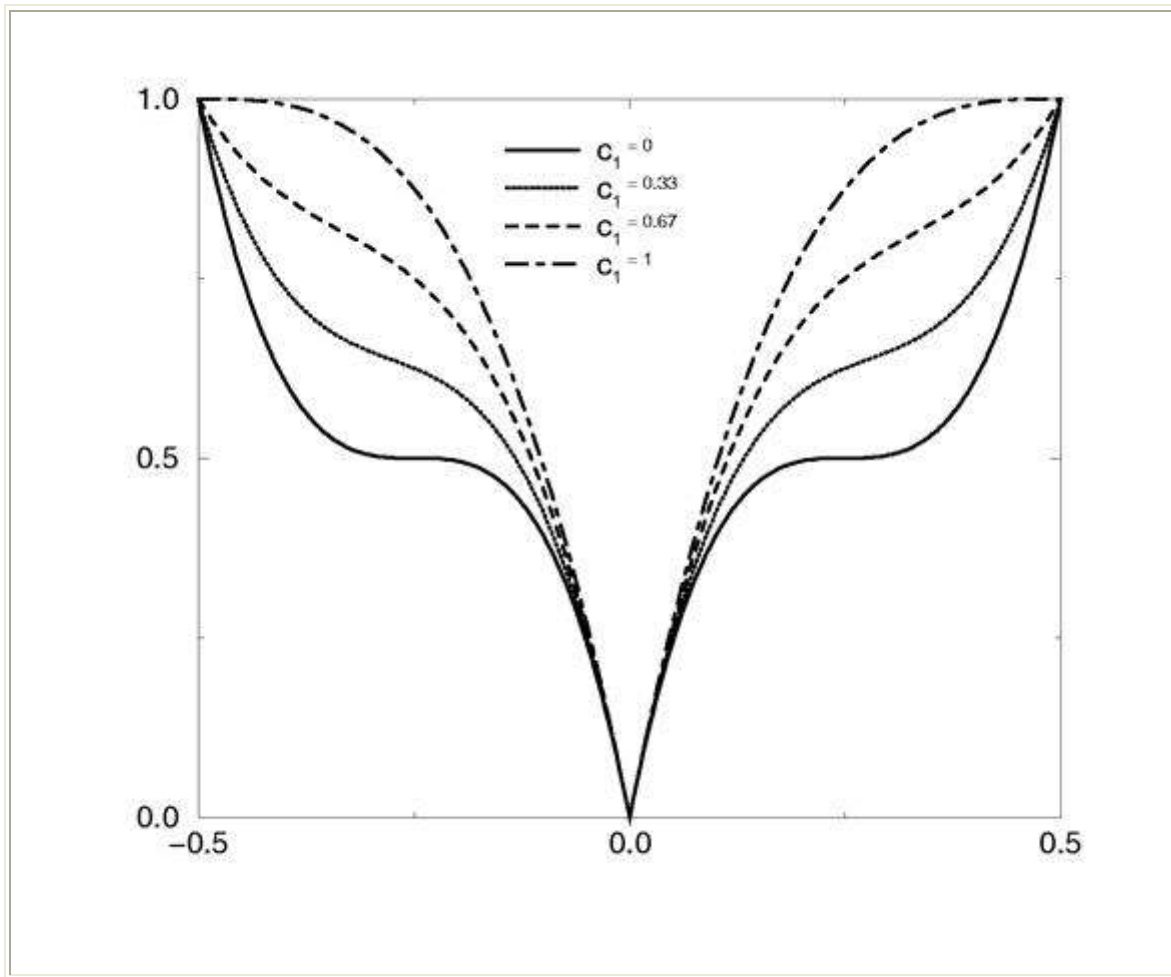



Fig. 227-Il pattern 'quilted' con c0=1 e diversi valori di c1.

Questa brusca pendenza può essere resa curva modificando i due valori di controllo. I valori di controllo modificano la pendenza in cima e in fondo alla curva. Un valore di 0 a entrambe le estremità darà una pendenza lineare, come è mostrato sopra, dando delle estremità spigolose. Un valore di 1 ad entrambe le estremità darà una curva ad "S" molto morbida e con le estremità arrotondate.

7.6.7.17 Radial (Radiale)

Il pattern `radial` consiste in una sfumatura radiale che si avvolge attorno all'asse y. Il colore corrispondente al valore 0.0 della mappa dei colori parte dall'asse delle x positive ed avvolge la mappa dei colori in direzione est - ovest con il colore di indice 0.25 in corrispondenza dell'asse -z, 0.5 sull'asse -x , 0.75 a +z e 1.0 di nuovo a +x. Normalmente questo pattern è usato con il modificatore `frequency` per creare più asce di colore attorno all'asse y. Il pattern `radial` usa l'onda `ramp_wave` di default, ma può usare qualunque tipo di onda. Il pattern `radial` può essere usato con mappe di colore, pigmento, normali, texture e slope.

7.6.7.18 Ripples (Increspature)

Il pattern `ripples` era originariamente ideato come pattern da usare per le normali. Rende la superficie simile a dell'acqua increspata. Le increspature partono da 10 punti casuali all'interno del cubo che si estende da <0,0,0> a <1,1,1>. Il pattern può essere quindi ridimensionato per avvicinare o distanziare i centri delle onde. Normalmente le increspature distano fra loto circa un'unità. La parola chiave `frequency` (frequenza) modifica lo spazio tra le increspature. La parola chiave `phase` può essere usata per farle scorrere verso l'esterno per ottenere realistiche animazioni. Vedi il

paragrafo "Number_Of_Waves (Numero d'Onde)" per maggiori dettagli. Quando viene usato per le normali, il pattern ripples fa uso di una funzione speciale di perturbazione delle normali e quindi non si può utilizzare con modificatori di mappe di normali, slope o di forme d'onda.
Quando viene usato in una frase pigment{...} o texture{...} il pattern ripples usa l'onda ramp_wave di default, ma può usare qualunque tipo di onda. Il pattern ripples può essere usato con mappe di colore, pigmento e texture.

7.6.7.19 Spiral1 (Spirale 1)

Il pattern spiral1 crea una spirale che si avvolge attorno all'asse delle y in modo simile ad una vite. La sintassi è :

```
pigment{
spiral1 NUMERO
}
```

Dove il valore di NUMERO determina quanti bracci della spirale si avvolgono sull'asse delle y. Il pattern spiral1 usa l'onda triangle_wave di default, ma può usare qualunque tipo di onda. Il pattern spiral1 può essere usato con mappe di colore, pigmento, normali, texture e slope.

7.6.7.20 Spiral2 (Spirale2)

Il pattern spiral2 è una versione modificata del pattern spiral1 con un aspetto straordinario. Il pattern spiral2 usa l'onda triangle_wave di default, ma può usare qualunque tipo di onda. Il pattern spiral2 può essere usato con mappe di colore, pigmento, normali, texture e slope.

7.6.7.21 Spotted

Il pattern spotted è identico al pattern bozo. Le prime versioni di POV-Ray non permettevano l'uso del modificatore turbulence con questo pattern. Ora che qualunque pattern può usare la turbolenza, non ci sono differenze tra bozo e spotted. Vedi il paragrafo "Bozo" per dettagli.

7.6.7.22 Waves (Onde)

Il pattern waves era originariamente concepito per l'uso come pattern di normali. Questo pattern è simile a ripples ma le onde sono più ampie ed arrotondate. L'effetto dà onde che assomigliano più ad onde marine che non a increspature sull'acqua. Le onde si irradiano da dieci posizioni casuali all'interno del cubo che si estende da <0,0,0> ad <1,1,1>. Ridimensionando il pattern si possono avvicinare od allontanare questi punti.
Normalmente le onde distano tra loro circa 1 unità. La parola chiave frequency modifica lo spazio tra le onde, mentre phase le fa scorrere verso l'esterno per ottenere realistici effetti di animazione.
Il numero di punti nei quali le onde hanno origine può essere cambiato con l'impostazione globale

```
gobal_settings{ number_of_waves DECIMALE}
```

che modifica i pattern ripples e waves presenti in tutta la scena. Vedi il paragrafo "Number_Of_Waves (Numero d'Onde)" per maggiori dettagli. Quando viene usato per le normali, il pattern waves fa uso di una funzione speciale di perturbazione delle normali e quindi non si può utilizzare con modificatori di mappe di normali, slope o di forme d'onda.
Quando viene usato in una frase pigment{...} o texture{...} il pattern waves usa l'onda ramp_wave di default, ma può usare qualunque tipo di onda. Il pattern waves può essere usato con mappe di colore, pigmento e texture.

7.6.7.23 Wood (Legno)

Il pattern wood consiste di cilindri concentrici centrati sull'asse delle z. Quando si usa un'appropriata mappa di colore, il pattern assomiglia alle venature degli anelli di crescita del legno. Con l'aggiunta di poca turbolenza, il pattern assume maggiore realismo, poiché per default wood non ha turbolenza.
A differenza della maggior parte dei pattern, wood usa un'onda triangolare (triangle_wave) per default. Ciò significa che la mappa dei colori viene ripetuta da 0.0 a 1.0 e poi al contrario, da 1.0 a 0.0. Ad ogni modo, si può usare qualunque forma d'onda. Il pattern può essere usato con mappe di colore, pigmento, texture, normali e slope.

7.6.7.24 Wrinkles (Spiegazzature)

Il pattern wrinkles era stato originariamente concepito come un pattern di normali. Fa uso di una funzione di rumore frattale del tipo 1/f simile a quella che usa granite, ma l'effetto è più netto. Il pattern può esssere usato per simulare ad esempio del cellophane spiegazzato, ma può dare anche un ottimo intonaco.
Quando wrinkles è usato come pattern di normali, fa uso di una funzione speciale di perturbazione delle normali e quindi non si può utilizzare con modificatori di mappe di normali, slope o di forme d'onda.
Quando viene usato in una frase pigment{...} o texture{...} il pattern wrinkles usa l'onda ramp_wave di default, ma può usare qualunque tipo di onda. Il pattern wrinkles può essere usato con mappe di colore, pigmento e texture.

7.6.8 Modificatori di Pattern

I modificatori di pattern sono frasi o parametri che alternano il calcolo o l'aspetto dei pattern. I modificatori color_map o pigment_map si applicano solo ai pigmenti, vedi il paragrafo "Pigmento". I modificatori bump_size, slope_map e normal_map si applicano solo alle normali, vedi il paragrafo "Normali". Il modificatore texture_map può essere applicato solo alle texture. Vedi il paragrafo "Mappe di Texture".
I modificatori di pattern descritti nel seguente paragrafo, possono essere usati con pigmenti, normali o pattern di texture.

7.6.8.1 Trasformare i Pattern.

I più comuni modificatori di pattern sono le trasformazioni translate, rotate, scale e matrix. Per dettagli su questi comandi vedere il paragrafo "Trasformazioni". Questi modificatori possono essere inseriti all'interno di frasi di descrizione di pigmenti, normali e texture per cambiarne la posizione, la dimensione e l'orientazione.
In generale, l'ordine delle trasformazioni relative ad altri modificatori di pattern come turbulence, color_map e altre mappe non è importante. Per esempio, scalare un pattern prima o dopo avere applicato la turbolenza non fa alcuna differenza. La turbolenza viene calcolata per prima, poi viene calcolato il ridimensionamento senza tenere conto dell'ordine delle due istruzioni. Comunque, per le frasi di warp l'ordine nel quale le trasformazioni sono attuate è importante. Vedi il paragrafo "Warp".

7.6.8.2 Frequenza e Fase

I modificatori frequency (frequenza) e phase (fase) si comportano come un particolare tipo dei modificatori scale e translate e si usano per le mappe di colore, di pigmento, di normali, texture e slope. In questo paragrafo useremo una mappa di colore come esempio, ma gli stessi principi possono essere applicati anche a tutte le altre mappe. La parola chiave frequency influisce sul numero di volte che una mappa viene ripetuta per ogni ciclo di un pattern. Per esempio,

`gradient` copre i valori della mappa dei colori che vanno da 0 ad 1 nello spazio che va da x=0 ad x=1. Aggiungendo la parola chiave `frequency 2.0` la mappa di colore verrà ripetuta due volte nello stesso spazio. Lo stesso effetto può essere ottenuto usando `scale 0.5*x`, quindi la parola chiave `frequency` non è indispensabile per pattern come i gradienti. Il pattern `radial` avvolge la mappa di colore intorno all'asse +y una sola volta. Se si desiderano due copie (o tre, o dieci, o cento...) della mappa, è necessario creare una mappa più grande. L'aggiunta della parola chiave `frequency 2.0` farà ripetere la mappa di colori due volte. Provate questo :

```
pigment {
radial
color_map{[0.5 color Red][0.5 color White]}
frequency 6
}
```

Il risultato è dato da sei insiemi di strisce radiali rosse e bianche sfalsate attorno all'oggetto. Il valore decimale dopo `frequency` può essere un valore qualunque. Valori maggiori di uno faranno sì che venga fatta più di una copia della mappa usata. Valori che vanno invece da 0.0 a 1.0 faranno sì che solo una frazione della mappa venga usata. Valori negativi invertono la mappa.
Il valore di `phase` causa uno spostamento degli elementi della mappa di colore in modo tale che la mappa inizi e finisca in una posizione diversa. Nell'esempio precedente, se vengono renderizzati fotogrammi successivi partendo da `phase 0.0`, seguito da `phase 0.1`, `phase 0.2` ecc. si creerà un'animazione nella quale le strisce ruotano. Lo stesso effetto si può ottenere facilmente ruotando il pigmento radiale usando la parola chiave `rotate y*angolo` ma ci sono altri casi in cui la parola chiave `phase` può essere indispensabile. Qualche volta capita di creare un bellissimo gradiente, o una bella mappa di colore per un legno, ma magari si vorrebbero sfasare leggermente i colori nella mappa. Potresti rimettere in ordine i vari elementi della mappa di colore, ma è un lavoro lungo. L'utilizzo di `phase` permette di 'spostare' tutti i colori mantenendoli in scala. Prova ad animare un pigmento `mandel` per vedere un effetto di rotazione della tavolozza dei colori.
Frequenza e fase non hanno effetto su pattern a blocchi come `checker`, `brick` ed `hexagon` e nemmeno su mappature di immagini, per pigmenti, normali e texture. Non hanno effetto neanche all'interno di frasi di descrizione delle normali, in associazione con `bumps`, `dents`, `quilted` o `wrinkles` poiché questi pattern di normali non possono fare uso di mappe di normali o slope. Possono essere usati con i pattern di normali `ripples` e `waves` anche se questi due pattern non possono fare uso di mappe di normali o slope. Quando sono usati con `ripples` e `waves`, `frequency` modifica lo spazio tra due onde successive e `phase` può avere valori da 0.0 ad 1.0 per fare muovere le onde in un'animazione. Questi valori si calcolano tramite la seguente formula :

```
nuovo_valore= fmod( vecchio*frequency+phase, 1.0)
```

7.6.8.3 Forma d'Onda

La maggior parte dei pattern che usano mappe di colore, pigmento, normali, texture o slope, utilizzano gli elementi all'interno della mappa nell'ordine da 0.0 a 1.0. I pattern `wood` e `marble` usano mappe che vanno da 0.0 ad 1.0 e poi le invertono e proseguono usando i colori da 1.0 a 0.0. La differenza può essere facilmente notata quando questi pattern sono usati come pattern di normali senza mappe.
I pattern come `gradient` o `onion` se usati nelle normali, renderanno la superficie ondulata, con onde che hanno una certa pendenza e quindi scendono bruscamente. Questo tipo di onda è detto `ramp_wave`. Comunque, la pendenza di `wood` e `marble` arriva ad un picco e poi scende usando un'onda triangolare. Nelle precedenti versioni di POV-Ray non c'era modo di cambiare i tipi di onda. Potevi simulare un'onda triangolare con una `ramp_wave` (*onda a dente di sega*) duplicando

gli elementi della mappa ed invertendoli, ma non c'era modo di usare un'onda a dente di sega su `wood` o su `marble`.
Ora, per un qualunque pattern che è definito da una mappa, si può modificare la forma d'onda usata. Per esempio,

```
pigment { wood color_map { Mappa} ramp_wave }
```

Sono disponibili anche onde sinusoidali (`sine_wave`) e 'scolpite' (`scallop_wave`). Questi tipi sono fra i più usati nei pattern di normali come una specie di mappa di pendenza (`slope_map`) incorporata. L'onda sinusoidale trasforma lo zigzag di un'onda a dente di sega in un'onda con transizioni di colore molto graduali. L'onda 'scallop' utilizza il valore assoluto dell'onda sinusoidale ed assomiglia a tela, se rimpicciolita, o a cilindri allineati, se ingrandita. Sebbene tutti questi tipi di onda possano essere utilizzati per pigmenti, normali o texture, le onde `sine_wave` e `scallop_wave` non sono altrettanto interessanti per i pigmenti o le texture come lo sono per le normali.
Le varie forme d'onda non hanno effetto su pattern a blocchi come `checker`, `brick` ed `hexagon` e nemmeno su mappature di immagini, mappe bump e di materiali. Non hanno effetto neanche all'interno di frasi di descrizione delle normali, in associazione con `bumps`, `dents`, `quilted` o `wrinkles` poiché questi pattern di normali non possono fare uso di mappe di normali o slope.

7.6.8.4 Turbolenza

La parola chiave `turbulence` seguita da un numero decimale o da un vettore può essere usata per 'agitare' qualunque pigmento, normale, texture, iridescenza o halo. Si possono usare diversi parametri facoltativi per controllare come viene calcolata la turbolenza. Per esempio:

```
pigment {
wood color_map { mia_mappa }
turbulence VETTORE
octaves DECIMALE
omega DECIMALE
lambda DECIMALE
}
```

I valori tipici per questa parola chiave sono compresi tra 0.0 (valore predefinito), che non aggiunge turbolenza e 1.0 o più, che aggiunge molta turbolenza. Se è specificato un vettore, saranno applicati valori diversi di turbolenza nelle direzioni x, y e z. Per esempio

```
turbulence <1.0, 0.6, 0.1>
```

ha una maggiore turbolenza nella direzione x, una quantità moderata nella direzione y e una piccolissima turbolenza nella direzione z.
La turbolenza usa una funzione casuale che aggiunge rumore, chiamata *Dnoise*. Questa funzione è simile a quella che aggiunge rumore al pattern `bozo` ma invece di restituire un singolo valore, calcola una *direzione*. Puoi immaginarla come la direzione nella quale soffia il vento. Punti vicini avranno valori simili, ma punti lontani avranno valori diversi in modo casuale.
In generale, l'ordine dei parametri della turbolenza relativo all'ordine dei modificatori di pattern come trasformazioni, mappe di colori o altre mappe, non è importante. Per esempio, scalare prima o dopo avere aggiunto la turbolenza non fa differenza. La turbolenza sarà calcolata per prima e poi verrà calcolato il ridimensionamento senza tenere conto di quale è stato specificato per primo. Vedi il paragrafo "Warp" per trovare un modo di aggirare questo problema. La turbolenza usa la funzione

DNoise per spostare un punto di alcuni passi detti *ottave* (`octaves`). Localizziamo il punto da calcolare, lo spostiamo di un po' usando la turbolenza per giungere ad un punto diverso, poi guardiamo il colore, o il pattern del nuovo punto. In effetti dice "non darmi il colore di questo punto...fai alcuni passi a caso in direzioni diverse e dammi *quel* colore". Ogni passo è generalmente lungo la metà di quello precedente.

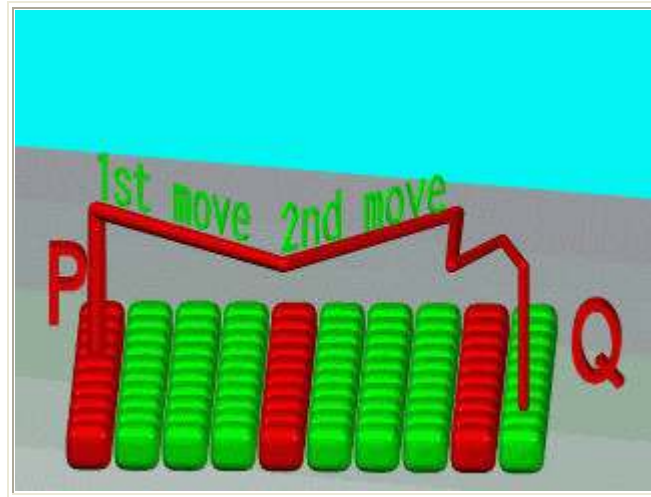



Fig. 228-Percorso casuale con turbolenza

L'ampiezza di questi 'passi' è controllata dal valore della turbolenza. Ci sono tre parametri aggiuntivi che controllano il calcolo della turbolenza. Sono `octaves`, `lambda` e `omega`. Sono tutti e tre facoltativi. Ciascuno di essi è seguito da un valore decimale e nessuno di essi ha effetto se non è insieme alla turbolenza.

7.6.8.5 Ottave (Octaves)

Il valore di `octaves` controlla il numero di passi che vengono eseguiti nel calcolo della turbolenza. Si possono usare valori compresi tra 1 e 10. Il valore di default di 6 è già sufficientemente alto ; non si noteranno grossi cambiamenti a valori superiori a sei dato che i passi successivi sono troppo piccoli. I valori decimali sono troncati a numeri interi. Valori piccoli di `octaves` danno una turbolenza più tenue ed ondulata e vengono calcolati velocemente, mentre valori più alti danno un effetto di turbolenza più marcato e richiedono un tempo di calcolo più lungo.

7.6.8.6 Lambda

Il parametro `lambda` controlla in maniera statistica quanto diverse sono le direzioni dei passi di calcolo della turbolenza. Il valore predefinito di 2.0 è abbastanza casuale. Valori vicini ad 1.0 modificano molto poco la direzione dei passi successivi, che tendono ad essere nella stessa direzione.

7.6.8.7 Omega

Il parametro `omega` controlla l'ampiezza relativa dei passi che vengono eseguiti nel calcolo della turbolenza. L'ampiezza di ogni passo è determinata moltiplicando `omega` per l'ampiezza del passo precedente. Il valore di default 0.5 significa che ogni passo (ottava) è lungo la metà del precedente. Alti valori di `omega` significano che la seconda terza, quarta e le ottave successive contribuiscono di più alla turbolenza, dando un aspetto più 'spigoloso', mentre valori più bassi danno una turbolenza dall'apparenza più sfumata.

7.6.8.8 Warp (Deformazioni)

La frase `warp{...}` è un modificatore di pattern simile alla turbolenza. La turbolenza agisce prendendo il punto in cui è calcolato il colore e spostandolo con una serie di passaggi casuali. Il modificatore `warp` invece, sposta il punto in un modo definito e specifico, non casualmente. La frase `warp{...}` supera alcune limitazioni della turbolenza tradizionale dando all'utente più controllo sull'ordine nel quale la turbolenza, le trasformazioni ed i modificatori di `warp` sono applicati al pattern.
Ci sono tre tipi di warp, ma la sintassi è stata pensata in modo da poterne aggiungere altri in futuro. I primi due, il 'warp di ripetizione' (*repeat warp*) ed il 'buco nero' (*black hole warp*) sono funzioni nuove per POV-Ray che modificano il pattern in modo geometrico. Il terzo tipo di warp è un metodo alternativo per specificare la turbolenza. La sintassi per usare una frase warp all'interno di una frase `pigment{...}` è la seguente :

```
pigment {
TIPO DI PATTERN
MODIFICATORI DEL PIGMENTO...
warp { FRASE WARP...}
ALTRI_MODIFICATORI...
}
```

Allo stesso modo, i warp possono essere usati nelle normali e nelle texture. Si possono avere tante frasi di warp quante se ne vogliono in ciascun pattern. La posizione della frase di warp rispetto agli altri modificatori come `color_map` o `turbulence` non è importante, ma la posizione delle frasi warp rispetto a loro stesse ed alle trasformazioni è importante. Laddove sono presenti più frasi warp e le relative trasformazioni, queste vengono calcolate nell'ordine in cui sono state specificate. Per esempio risultati possono essere diversi se si applica prima una traslazione e poi warp o viceversa.

7.6.8.8.1 Buco Nero

Il warp '*buco nero*' si chiama così perché è simile ai veri buchi neri. Così come questi, non è realmente possibile vederlo. L'unico modo per intuirne la presenza è quello di osservare l'effetto che ha sugli oggetti che lo circondano. Diversamente da un vero buco nero, non può inghiottirti e comprimerti a dimensioni, diciamo, di 10-20 micron di diametro (ipotesi molto ottimistica, N.d.T.) se ti avvicini troppo (su questa parte stiamo ancora lavorando ☺). Prendiamo per esempio le venature di una texture legno. Usando la normale turbolenza di POV-Ray ed altri modificatori di texture, possiamo ottenere delle belle venature, ma nella sua casualità, la texture è regolare, è 'regolarmente casuale' ! Aggiungendo un 'black hole' creeremo un disturbo localizzato in una o più posizioni. Il buco nero può avere l'effetto sia di 'risucchiare' la texture che lo circonda, sia di respingerla. Nel secondo caso, se applicato ad un legno, può simulare i nodi delle venature. In questo esempio parleremo di una texture 'legno' perché è l'ideale per spiegare il funzionamento di `black_hole`. Ad ogni modo, questa funzione può essere usata in qualunque tipo di texture. L'effetto che il 'buco nero' ha sulla texture può essere specificato. Per default risucchia il colore con una forza calcolata esponenzialmente (inverso del quadrato esattamente come l'attrazione gravitazionale), ma può essere cambiata. I buchi neri possono essere usati ovunque sia permessa una frase `warp{...}`. La sintassi è :

```
warp
{
black_hole <CENTER>, RADIUS
[falloff VALUE]
[strength VALUE]
[repeat <VECTOR>]
[turbulence <VECTOR>]
```

```
[inverse]
}
```

Alcuni esempi sono...

```
warp
{
black_hole <0, 0, 0>, 0.5
}

warp
{
black_hole <0.15, 0.125, 0>, 0.5
falloff 7
strength 1.0
repeat <1.25, 1.25, 0>
turbulence <0.25, 0.25, 0>
inverse
}

warp
{
black_hole <0, 0, 0>, 1.0
falloff 2
strength 2
inverse
}
```

Per capire completamente come funziona `black_hole` è importante conoscerne le basi teoriche. Un buco nero (o qualunque warp) funziona prendendo un punto e spostandolo in un'altra posizione. La quantità di perturbazione dipende dalla forza del buco nero nel punto in cui si applica ed è direttamente connessa alla quantità di spostamento che vedi avvenire nella texture. Più forte è il buco nero alle coordinate del punto su cui agisce, più questo sarà spostato. Lo spostamento avviene sempre lungo il vettore che collega il centro del buco nero ed il punto sul quale influisce. I buchi neri si suppongono sferici. Perché un punto subisca l'effetto del buco nero, deve trovarsi all'interno del volume della sfera. Supponendo di avere un buco nero a <1,1,1> ed un punto a <1,2,1>, se questo punto è perturbato di una quantità totale di +1 unità, la sua nuova posizione sarà <1,3,1>, che si trova sulla congiungente i punti <1,1,1> ed <1,2,1>. In questo caso il punto è allontanato dal buco nero ; questo non è il suo normale comportamento, ma è utile per un esempio. I parametri di un buco nero sono i seguenti :

Centro il centro del buco nero
Raggio il suo raggio
Falloff la potenza di 2 con la quale l'effetto decade (default 2)
Strenght la quantità della trasformazione
Inverted se impostato allontana i punti invece di risucchiarli
Repeat se impostato, avremo molti buchi neri invece di uno
Turbulence se impostato, la posizione di ognuno dei buchi neri (se viene usato `repeat`) è casuale.
Repeat_Vector il vettore <x,y,z> lungo il quale ripetere i warp.

Turbulence_Vector il valore massimo in tre dimensioni di turbolenza casuale.

Ciascuno di questi parametri viene discusso di seguito :

Centro un vettore che definisce il centro della sfera che rappresenta il buco nero.
Raggio un numero che indica la lunghezza in unità del raggio della sfera
Se un punto non è all'interno del raggio non sarà influenzato dalla presenza del buco nero.

Falloff la potenza di decadimento dell'effetto. Il valore predefinito è 2. La forza del punto nero in un punto qualunque, prima di applicare il modificatore della forza, è come segue.
Primo, converti la distanza dal centro ad un valore da 0 a 1 proporzionale al raggio. Un punto sulla superficie della sfera avrà valore 0, un punto al centro avrà valore 1.
Chiamiamo questo valore *c* ed eleviamolo alla potenza specificata da `falloff`. Per default `falloff` è 2, quindi otteniamo *c*^2 (*c* al quadrato). Il valore che otteniamo è la forza del buco nero in quella posizione che, dopo aver applicato il fattore `strenght` determina la perturbazione del punto nello spazio. Ad esempio se *c* è 0.5 la forza è 0.5^2=0.25. Se *c* è 0.25 la forza è 0.125. Ma se *c* è esattamente uguale ad 1, la forza è 1. Usando la potenza di 2 possiamo vedere che via via che *c* diminuisce la forza cala in modo esponenziale con un decadimento geometrico (secondo la potenza di -2). In parole povere la forza è molto più forte vicino al centro di quanto non lo sia verso l'esterno. Aumentando `falloff` si può aumentare questa differenza. Un valore alto significa che i punti sulla superficie della sfera saranno appena influenzati e che quelli vicini al centro saranno fortemente perturbati. Un valore di 1 per il `falloff` significa che l'effetto è lineare. Un punto che si trovi a metà strada tra il centro e la superficie subirà una forza uguale a 0.5. Un valore di `falloff` tra 0 e 1 significa che avvicinandosi alla superficie la forza aumenta invece di diminuire. Questo può avere qualche utilità ma ha un effetto collaterale : l'effetto di un buco nero cessa al di fuori del suo perimetro, questo significa che i punti appena all'interno del perimetro saranno fortemente perturbati, mentre quelli appena fuori non saranno perturbati e questo darebbe un bordo visibile di forma sferica. Un valore di `falloff` uguale a 0 significherebbe che la forza sarebbe uguale a 1 per tutti i punti all'interno del buco nero poiché ogni numero diverso da zero elevato alla potenza 0 dà 1.
La dimensione dello spostamento del punto è data fondamentalmente dal valore della forza dopo il ridimensionamento. Facciamo un esempio : supponiamo di avere un buco nero di raggio 2 ed un punto ad un unità dal centro, questo significa che si trova esattamente tra il centro e la superficie e che *c* sarebbe esattamente 0.5. Se usiamo il valore di 2 per `falloff` la forza su quel punto sarebbe 0.5^2=0.25. Questo significa che dobbiamo spostare questo punto della sua distanza dal centro moltiplicata per 0.25, in questo caso la distanza è di una unità e quindi spostiamo il punto di 1*0.25=0.25 unità. Questo ci dà una distanza dal centro di 1.0-(1.0*0.25)=0.75 unità dal centro, sulla congiungente dal centro alla posizione originaria del punto. Se i punti fossero parte, diciamo, della venatura di un legno, questa venatura sembrerebbe convergere verso il centro (invisibile) del buco nero. Se fosse impostata la parola chiave `inverse` sarebbe invece respinta dal buco nero e la posizione finale del punto sarebbe 1.0+(1.0*0.25)=1.25 sulla retta congiungente il centro con la posizione originaria del punto.

Strenght (forza). La forza da un po' più di controllo su quanto un punto viene perturbato dal buco nero. Fondamentalmente la forza del buco nero come calcolata sopra viene moltiplicata per il valore di `strenght` che di default è 1.0. Se imposti `strenght` a 0.5 per esempio, tutti i punti all'interno del buco nero verranno spostati solo della metà di quanto sarebbero spostati normalmente. Se imposti `strenght` a due verranno spostati del doppio. C'è una limitazione all'ultimo esempio : lo spostamento può essere al massimo la distanza originaria dal centro del buco nero. Questo significa che un punto lontano 0.75 unità dal centro può essere spostato di un massimo di 0.75 unità, indipendentemente dal valore di `strenght`. Il risultato di questa limitazione è che si

avrà una zona vicino al centro del buco nero in cui tutti i punti il cui valore finale della forza era maggiore o uguale a uno vengono spostati.

Inverted se si inserisce la parola chiave `inverted` i punti sono respinti dal centro invece di venirne attratti.

Repeat : `repeat` permette di simulare l'effetto di molti buchi neri senza doverli dichiarare esplicitamente. `Repeat` è un vettore che dica a POV-Ray di usare il buco nero definito in più posizioni. Se non ti interessa la teoria dietro a tutto questo salta il testo seguente e usa i valori riassunti più avanti (non osate saltarlo, altrimenti perché abbiamo fatto la fatica di tradurlo ! N.d.T.). Usare `repeat` divide la scena in parallelepipedi, il primo dei quali si estende dall'origine al punto specificato dal vettore `repeat`. Supponiamo che il vettore sia <1,5,2>. Il primo parallelepipedo si estenderebbe tra <0,0,0> e <1,5,2>. Questo parallelepipedo si ripete quindi ce ne sarebbero tra <0,0,0> e <-1,-5,-2>, tra <1,5,2> e <2,10,4> e così via in tutte le direzioni all'infinito. Quando usi `repeat` il vettore che specifica il centro del buco nero non specifica una posizione assoluta rispetto agli assi, ma relativa ad ogni parallelepipedo. Si possono usare solo valori positivi, valori negativi darebbero risultati indefiniti. Supponiamo che il centro sia <0.5,1,0.25> e che il vettore `repeat` sia <2,2,2>. Questo ci dà un parallelepipedo tra <0,0,0> e <2,2,2> ecc. i centri dei buchi neri per questi blocchi sarebbero : <0,0,0>+<0.5,1,0.25>=<0.5,1,0.25> e <2,2,2>+<0.5,1,0.25>=<2.5,3.0,2.25>. A causa del metodo con cui vengono eseguiti i calcoli c'è una restrizione sui valori che si possono specificare per il vettore `repeat`. Fondamentalmente, ogni buco nero deve essere racchiuso totalmente all'interno del rispettivo parallelepipedo, senza incrociare le pareti di quelli adiacenti. Questo significa che in tutte e tre le dimensioni, la posizione del centro non deve essere minore del raggio e che il valore del vettore `repeat` per ogni dimensione deve essere maggiore o uguale della somma di raggio e posizione del centro, poiché altrimenti permetterebbe ad un buco nero di incrociare le pareti di un parallelepipedo. Semplicemente, per x, y e z deve essere

```
raggio <= centro <= repeat - raggio
```

Se il vettore `repeat` è troppo piccolo per soddisfare questo criterio, verrà aumentato e sarà visualizzato un messaggio di avvertimento. Se la posizione del centro è minore del raggio, esso verrà spostato, ma non verranno visualizzati messaggi di avvertimento.
Nota che nessuna delle regole viste sopra significa che non puoi sovrapporre i buchi neri. Questo è possibile ed anzi può essere molto utile. La restrizione si applica solo ad elementi *dello stesso buco nero* che viene ripetuto. Puoi dichiarare un secondo buco nero che si ripete e i cui elementi possono tranquillamente sovrapporsi a quelli del primo causando le interazioni appropriate.
Si può attribuire il valore zero a una qualunque delle dimensioni del vettore `repeat`, col significato che il buco nero non sarà ripetuto in quella direzione.

Turbulence. La turbolenza può essere utilizzata solo con `repeat`. Permette di inserire un elemento di casualità nel metodo con cui i buchi neri si ripetono, per dar loro un aspetto più naturale. Un buon esempio sono i nodi nel legno : sarebbe artificioso se i nodi si ripetessero tutti ala stessa distanza.
Il vettore `turbulence` viene sommato alla posizione del centro di ogni buco nero, dopo che ogni sua coordinata viene moltiplicata per un valore casuale che va da 0 ad 1.
Ad esempio, supponi che un `black_hole` sia a <2,2,2>. Hai specificato un vettore turbolenza di <4,5,3>, che significa che non vuoi che la posizione del buco nero possa variare più di quattro unità nella direzione delle x, cinque nella direzione delle y e tre nella direzione delle z. Le possibili nuove posizioni sono le seguenti :

x può variare tra 2 e 6

y può variare tra 2 e 7
z può variare tra 2 e 5

La posizione del centro del buco nero che ne risulta è casuale (ma coerente, in modo che i rendering possano essere ripetibili) e posta all'interno di quell'intervallo.
C'è una limitazione nell'uso della turbolenza, che riflette quella sull'uso del vettore `repeat`. Non si può specificare un valore che farebbe uscire un buco nero dal suo parallelepipedo. Poiché POV-Ray non sa in anticipo di quanto verrà spostato il buco nero a causa della natura casuale dello spostamento, inserisce una regola simile a quella valida per `repeat`, tranne che aggiunge la variazione massima possibile al centro. Ad esempio, supponi di avere un buco nero centrato ad <1.0,1.0,1.0>, di raggio 0. 5 e turbolenza <0.5,0.25,0>. Normalmente, il valore minimo di `repeat` sarebbe <1.5,1.5,1.5>, ma se teniamo conto della turbolenza, il minimo diventa <2.0,1.75,1.5>.

Riassunto di Repeat : per tutti e tre gli assi x, y, z il centro deve essere spostato almeno del raggio ed il valore del vettore `repeat` deve essere almeno maggiore della somma di centro, raggio e turbolenza. L'eccezione è che `repeat` può essere 0, col significato che non ci sono ripetizioni su nessuno degli assi.

7.6.8.8.2 Ripetizione

Il warp 'ripetizione' (*repeat*) fa sì che una parte del pattern venga ripetuta all'infinito. Taglia una 'fetta' del pattern e ne crea copie multiple affiancate. Questo tipo di warp ha molti usi, ma è stato originariamente concepito per riprodurre l'effetto tipico dell'impiallacciatura. L'impiallacciatura si ottiene ricavando fette di legno molto sottili da un tronco e incollandole su un altro materiale di supporto. L'effetto è quello di un motivo di anelli affiancati, ma ogni componente è in realtà spesso circa 8/10 di millimetro.
La sintassi per il warp `repeat` è la seguente :

```
warp { repeat VETTORE offset VETTORE flip VETTORE }
```

Il vettore `repeat` specifica la direzione in cui il vettore si ripete e la larghezza dell'area che viene ripetuta. Questo vettore deve essere *su un asse*. In altre parole, due delle sue tre componenti devono essere uguali a zero. Ad esempio :

```
pigment {
wood
warp {repeat 2*x}
}
```

significa che da x=0 ad x=2 hai il pattern come è normalmente, ma da x=2 a x=4 ottieni lo stesso pattern, spostato di due unità nella direzione +x. Sfortunatamente si ottengono copie esatte e ciò non è molto realistico. Il vettore `offset`, facoltativo, trasla il pattern ogni volta che esso si ripete. Per esempio:

```
pigment {
wood
warp {repeat x*2 offset z*0.05}
}
```

significa che otteniamo la prima copia del pattern, da x=0 a x=2, con il pattern preso a z=0, ma tra x=2 e x=4 POV-Ray lo calcola per z=0.05. Nell'intervallo da x=4 a x=6 viene calcolato per z=0.10 ecc. In questo modo ogni copia del pattern è leggermente diversa dalla precedente. Non ci sono

restrizioni sul vettore `offset`.
Infine, il vettore `flip` (*capovolgi*) fa ribaltare il pattern nella direzione specificata. La prima copia del pattern nella direzione positiva non è ribaltata, mentre della seconda e di tutte le copie 'pari' otterremo l'immagine speculare. Il vettore `flip` è un vettore a tre componenti booleane, cioè ogni componente specifica se il pattern deve essere capovolto lungo un dato asse. Per esempio :

```
pigment {
wood
warp {repeat 2*x flip <1,1,0>}
}
```

significa che ogni copia 'pari' del pattern sarà capovolta rispetto agli assi x e y ma non rispetto all'asse z. Tutti i valori diversi da zero significano che il pattern deve essere capovolto.

7.6.8.8.3 Turbolenza

Il linguaggio di POV-Ray contiene un'ambiguità sul modo in cui possono essere specificate la turbolenza e le trasformazioni come traslazione, rotazione e ridimensionamento. Normalmente la turbolenza viene calcolata per prima. Poi, tutte le altre trasformazioni vengono eseguite indipendentemente dall'ordine con cui vengono specificate rispetto alla turbolenza. Per esempio, questa frase

```
pigment {
wood
scale .5
turbulence .2
}
```

ha esattamente lo stesso significato di

```
pigment {
wood
turbulence .2
scale .5
}
```

La turbolenza viene calcolata per prima. Un esempio migliore si ha con la turbolenza lungo un asse insieme alle rotazioni : confronta infatti

```
pigment {
wood
turbulence 0.5*y
rotate z*60
}
```

con

```
pigment {
wood
rotate z*60
turbulence 0.5*y
}
```

Il risultato sarà lo stesso in entrambi i casi anche se sembra che le due istruzioni dovrebbero dare risultati diversi.
Questo comportamento non si può cambiare perché altrimenti tutte le scene scritte per le versioni precedenti di POV-Ray, in cui non ci sono differenze nell'ordine turbolenza - trasformazioni, sarebbero renderizzate in modo diverso.
Invece, specificando la turbolenza all'interno di una frase `warp{...}`, si permette a POV-Ray di distinguere l'ordine in cui turbolenza e trasformazioni compaiono. Questo è un esempio di un 'warp turbolenza' :

```
warp { turbulence <0,1,1> octaves 3 lambda 1.5 omega 0.3 }
```

Il significato di questa frase è che :

```
pigment {
wood
translate <1,2,3> rotate x*45 scale 2
warp { turbulence <0,1,1> octaves 3 lambda 1.5 omega 0.3 }
}
```

produce un risultato **diverso** da :

```
pigment {
wood
warp { turbulence <0,1,1> octaves 3 lambda 1.5 omega 0.3 }
translate <1,2,3> rotate x*45 scale 2
}
```

La turbolenza può essere specificata senza inserirla in una frase `warp{...}`, ma in questo modo non si può controllare l'ordine in cui turbolenza e trasformazioni vengono calcolate.
Le regole di calcolo sono le seguenti :

1) Prima viene applicata la turbolenza *non inclusa* in frasi `warp{...}`, indipendentemente dall'ordine con cui compare rispetto alle trasformazioni o ad altri warp.
2) Vengono applicate le trasformazioni una dopo l'altra, comprese le frasi `warp{...}` *nell'ordine specificato dall'utente*. Quindi, se vuoi che la turbolenza sia valutata *in sequenza* con le altre istruzioni, devi specificarla all'interno di una frase `warp{...}` nell'ordine giusto.

7.6.8.9 Modificatori di Immagini Bitmap

Un modificatore di immagine bitmap è un modificatore utilizzato in frasi `image_map`, `bump_map` o `material_map` per specificare come l'immagine in due dimensioni deve essere applicata (mappata) alla superficie tridimensionale. Alcuni modificatori si applicano a particolari tipi di mappatura e sono trattati in paragrafi a parte. I modificatori trattati nei seguenti paragrafi si possono applicare ai tipi di mappatura elencati sopra.

7.6.8.9.1 L'Opzione 'ONCE'

Normalmente applicando una mappatura si creano infinite copie dell'immagine, quadrate di lato unitario, che ricoprono l'intero piano x-y, come piastrelle. Aggiungendo la parola chiave `once` dopo il nome del file puoi eliminare tutte le copie della mappa tranne quella che si trova nel riquadro che va da (0,0) a (1,1). In mappature di immagini le zone al di fuori di questo riquadro sono completamente trasparenti. Nelle mappature di normali alle aree esterne non sono applicate

normali.
Nelle mappature di materiali alle aree esterne è applicata la prima texture della lista.
Per esempio :

```
image_map {
gif "mypic.gif"
once
}
```

7.6.8.9.2 L'Opzione 'MAP_TYPE'

A meno che non si specifichi in modo diverso, l'immagine sarà proiettata sul piano x-y, con un metodo che è detto **mappatura planare**. Questa opzione può essere modificata aggiungendo la parola chiave `map_type` seguita da un numero che specifica il metodo con il quale l'immagine circonderà l'oggetto.

`map_type 0` dà la mappatura predefinita e quindi planare già ampiamente descritta.

`map_type 1` dà una mappatura sferica. Assume che l'oggetto sia una sfera di qualunque dimensione posta nell'origine. L'asse y unisce i poli sud e nord della sfera. La cima e il fondo dell'immagine toccano i due poli senza tenere conto delle dimensioni precedenti dell'immagine. La parte sinistra dell'immagine inizia dal lato dell'asse x positivo e trascina l'immagine lungo una rotazione intorno all'asse y. L'immagine copre la sfera esattamente una volta. La parola chiave `once` non ha nessun rapporto con questi comandi.

Con `map_type 2` si avrà una mappatura cilindrica. Si assume che un cilindro di un qualunque diametro giaccia lungo l'asse y. L'immagine si srotola intorno al cilindro con le stesse modalità viste per la sfera e rimane alta 1 cioè da y=0 a y=1. Questo è applicato a tutte le altezze superiori a meno che non sia specificata la parola chiave `once`.

Infine `map_type 5` è a forma di toro. Si assume cioè che un toro di raggio maggiore uguale ad uno sia posizionato all'origine del piano x-z. L'immagine si comporta come nei due metodi descritti sopra. La parte superiore ed inferiore del contorno della mappa si muoveranno attorno al toro e si uniranno nella parte interna del toro.

I tipi 3 e 4 sono ancora in via di ampliamento.

Per esempio :

```
sphere{<0,0,0>,1
pigment{
image_map {
gif "world.gif"
map_type 1
}
}
}
```

7.6.8.9.3 L'Opzione 'INTERPOLATE'

Aggiungendo la parola chiave `interpolate` è possibile smussare l'aspetto irregolare di un'immagine. Quando POV-Ray richiede un valore di colore per una mappatura di un'immagine, o di normali, spesso cerca un punto che non corrisponde direttamente ad un pixel della mappa, ma che

si trova in mezzo tra due pixel colorati diversamente. L'interpolazione restituisce un valore medio in modo tale che i passaggi tra i pixel di un'immagine siano più fluidi.
Sebbene sia possibile usare `interpolate` in mappe di materiali, la mappa viene interpolata *prima* che vengano assegnate le texture. Non viene interpolato il colore finale come invece si potrebbe sperare. In generale, l'interpolazione di una mappa dei materiali non è di particolare utilità, ma ssarà migliorata in futuro.
Al momento sono disponibili due tipi di interpolazione : `interpolate 2` (bilineare) e `interpolate 4` (distanza normalizzata). Ad esempio,

```
image_map {
gif "mypic.gif"
interpolate 2
}
```

Nessuna interpolazione viene effettuata a meno che non vengano specificati questi parametri. La più veloce delle due opzioni è `interpolate 4`, ma il metodo bilineare è più funzionale per questo scopo. Normalmente viene utilizzato proprio questo.
Se la tua mappa appare irregolare, prova ad usare l'interpolazione invece di aumentare la risoluzione dell'immagine ed il risultato può essere ottimo.

7.7 Effetti Atmosferici

Gli effetti atmosferici sono un gruppo di funzioni che influenzano lo sfondo e l'atmosfera nella quale si trova la scena. POV-Ray permette di renderizzare diversi effetti atmosferici quali la nebbia,

7.7.1 Atmosfera

Annotazione importante : la funzione atmosfera è sperimentale per POV-Ray 3.0. C'è un'alta probabilità che questa funzione venga cambiata in futuro. Non possiamo garantire che le scene che usano l'atmosfera nella versione 3.0 appariranno identiche nelle versioni future o che rimanga la piena compatibilità con la sintassi attuale.
Le immagini generate al computer normalmente sono ambientate in uno spazio vuoto che non permette il rendering di fenomeni naturali come il fumo, i raggi di luce ecc. Un metodo molto semplice per aggiungere la nebbia alla scena è descritto nel paragrafo "Nebbia". Questo tipo di nebbia non interagisce con alcuna luce. Non mostrerà i raggi luminosi o altri effetti e quindi non è molto realistico.
L'atmosfera supera alcune delle limitazioni della nebbia calcolando le interazioni tra la luce e le particelle presenti nell'atmosfera usando un campionamento sul volume. Quindi fasci di luce diventeranno visibili e gli oggetti proietteranno ombre *nel* fumo o nella nebbia.
La sintassi per l'atmosfera è :

```
atmosphere {
type TIPO
distance DISTANZA
[ scattering DISPERSIONE ]
[ eccentricity ECCENTRICITA' ]
[ samples CAMPIONI ]
[ jitter JITTER ]
[ aa_threshold SOGLIA ]
[ aa_level LIVELLO ]
[ colour <COLORE> ]
}
```

La parola chiave `type` determina il metodo di dispersione che deve essere usato. Ci sono cinque differenti funzioni di fase che rappresentano i diversi modelli : isotropo, di Rayleigh, di Mie (nebbia) e di Henyey-Greenstein.
La dispersione isotropa è la forma più semplice di dispersione perché è indipendente dalla direzione. La quantità di luce dispersa dalle particelle dell'atmosfera non dipende dall'angolo tra la direzione della vista e la luce incidente.
La dispersione di Rayleigh è un modello di dispersione per particelle molto piccole come le molecole dell'aria. La quantità di luce dispersa dipende dall'angolo di incidenza della luce. E' più grande quando la luce incidente è parallela alla direzione della vista e più piccola quando la luce incidente è perpendicolare. Il modello di Rayleigh usato da POV-Ray non prende in considerazione la dipendenza della dispersione dalla lunghezza d'onda della luce.

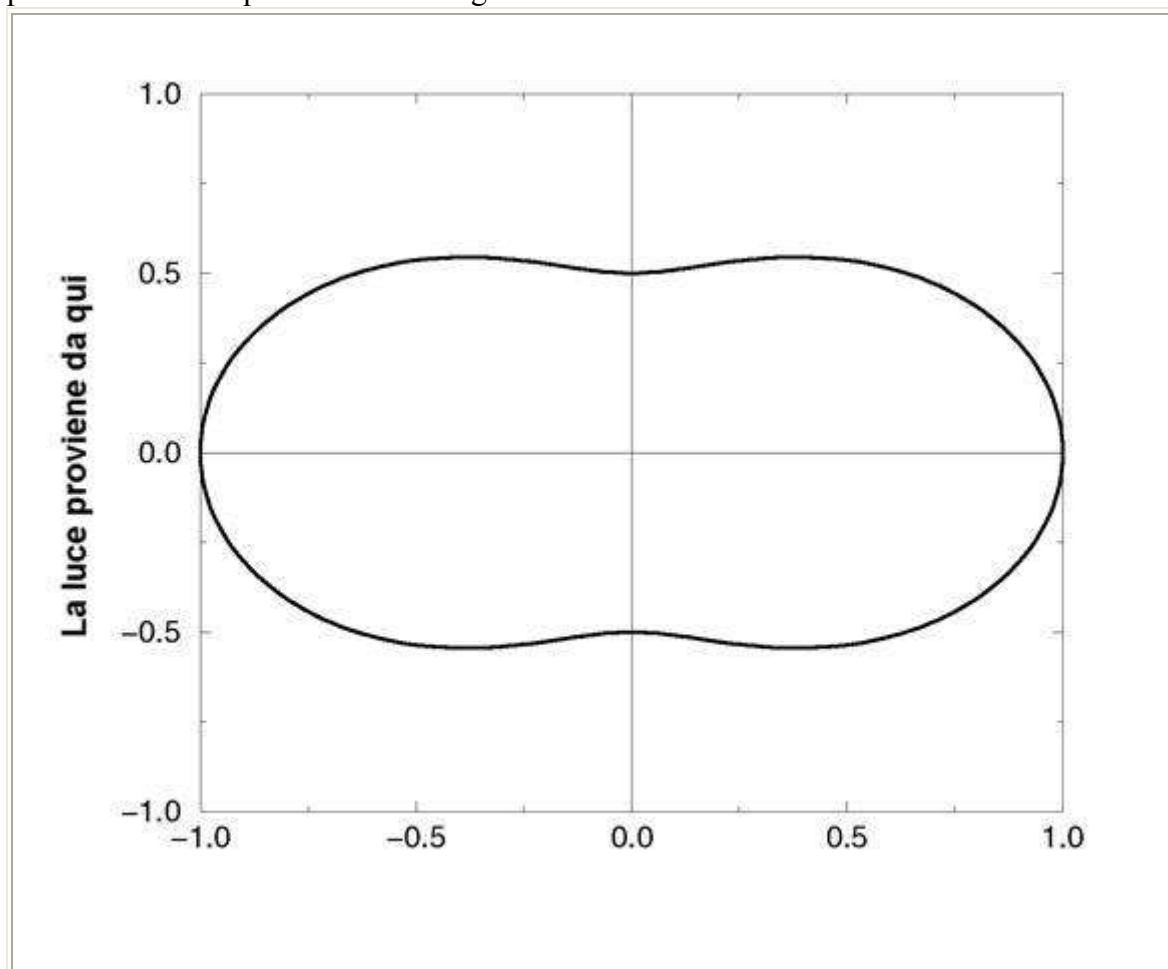



Figura 229- La funzione di dispersione di Rayleigh

La dispersione di Mie è usata per particelle abbastanza piccole come le minuscole gocce d'acqua che costituiscono la nebbia, le nuvole e le particelle inquinanti. Questo modello di dispersione è estremamente direzionale ; la quantità di luce dispersa è maggiore quando la luce incidente è rivolta direttamente contro l'osservatore. E' minore quando la luce incidente si allontana dall'osservatore, rimanendo parallela con la sua vista. C'è inoltre differenza tra i modelli che descrivono la foschia e l'atmosfera inquinata. Il modello che descrive questo secondo tipo di atmosfera è molto più direzionale dell'altro.

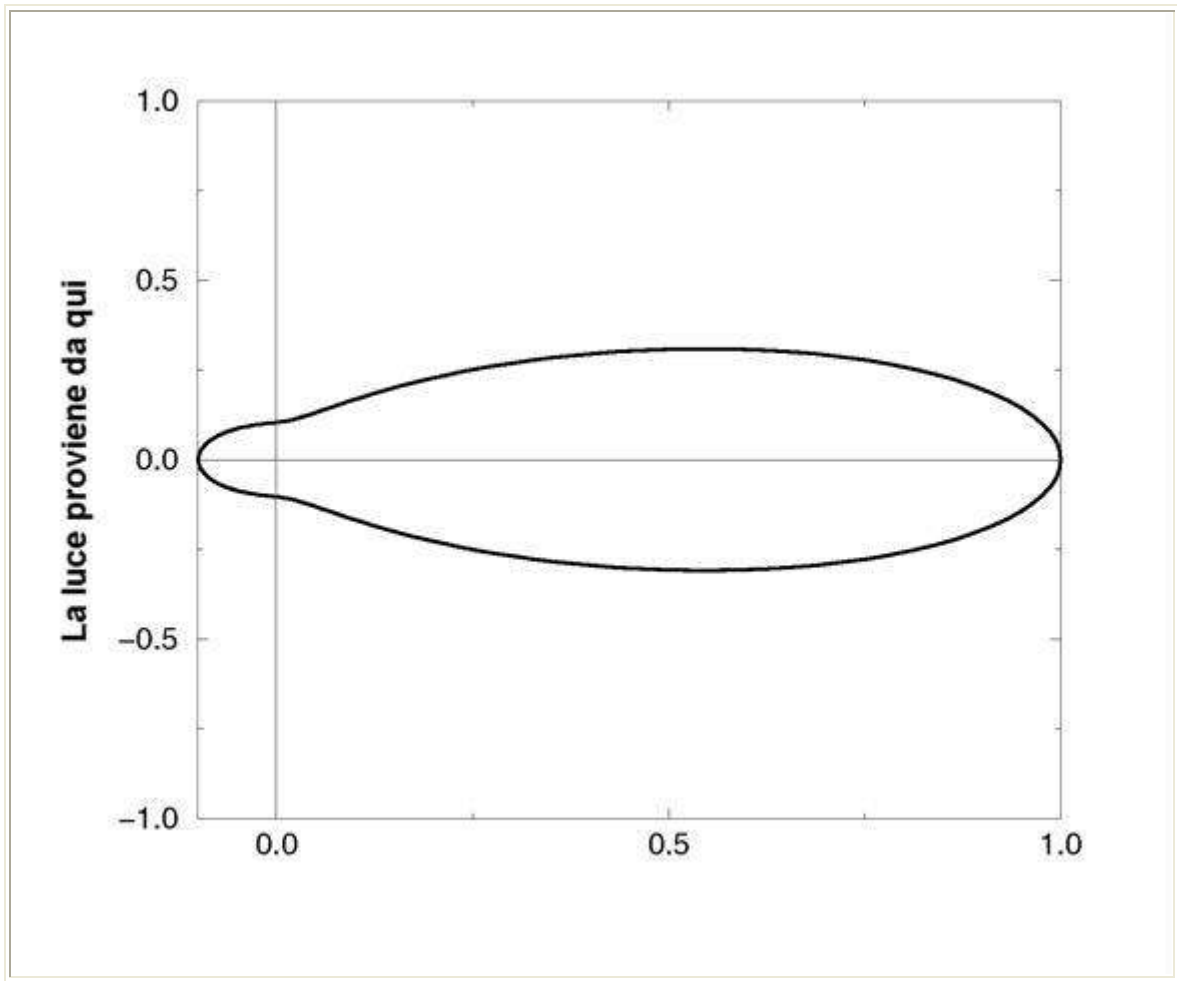



Figura 230- La funzione di dispersione di Mie per la foschia

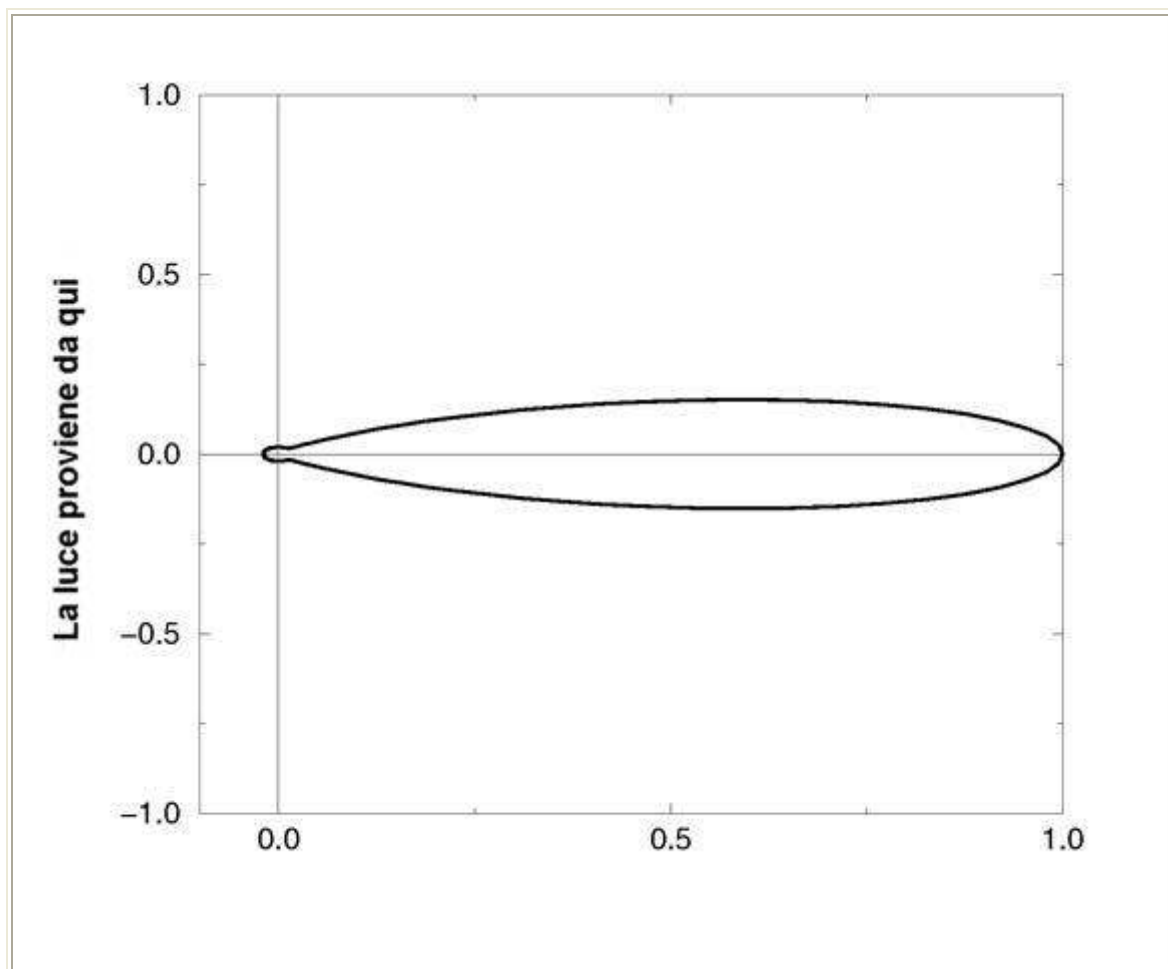



Figura 231- La funzione di dispersione di Mie per particelle più grandi (aria inquinata)

La dispersione di Heyney-Greenstein è basata su una funzione analitica e può essere usata per modellare una grande varietà di tipi di dispersioni. La funzione modella un'ellisse con una determinata eccentricità, e. Quest'eccentricità è specificata dalla parola chiave `eccentricity` che è usata solo per questo tipo di atmosfera (tipo 5). Il valore 0 assegnato all'eccentricità definisce la dispersione isotropa, mentre valori positivi produrranno una dispersione nella direzione della luce e valori negativi produrranno dispersione nella direzione opposta alla luce. Valori molto grandi di e incrementeranno la direzionalità della dispersione.

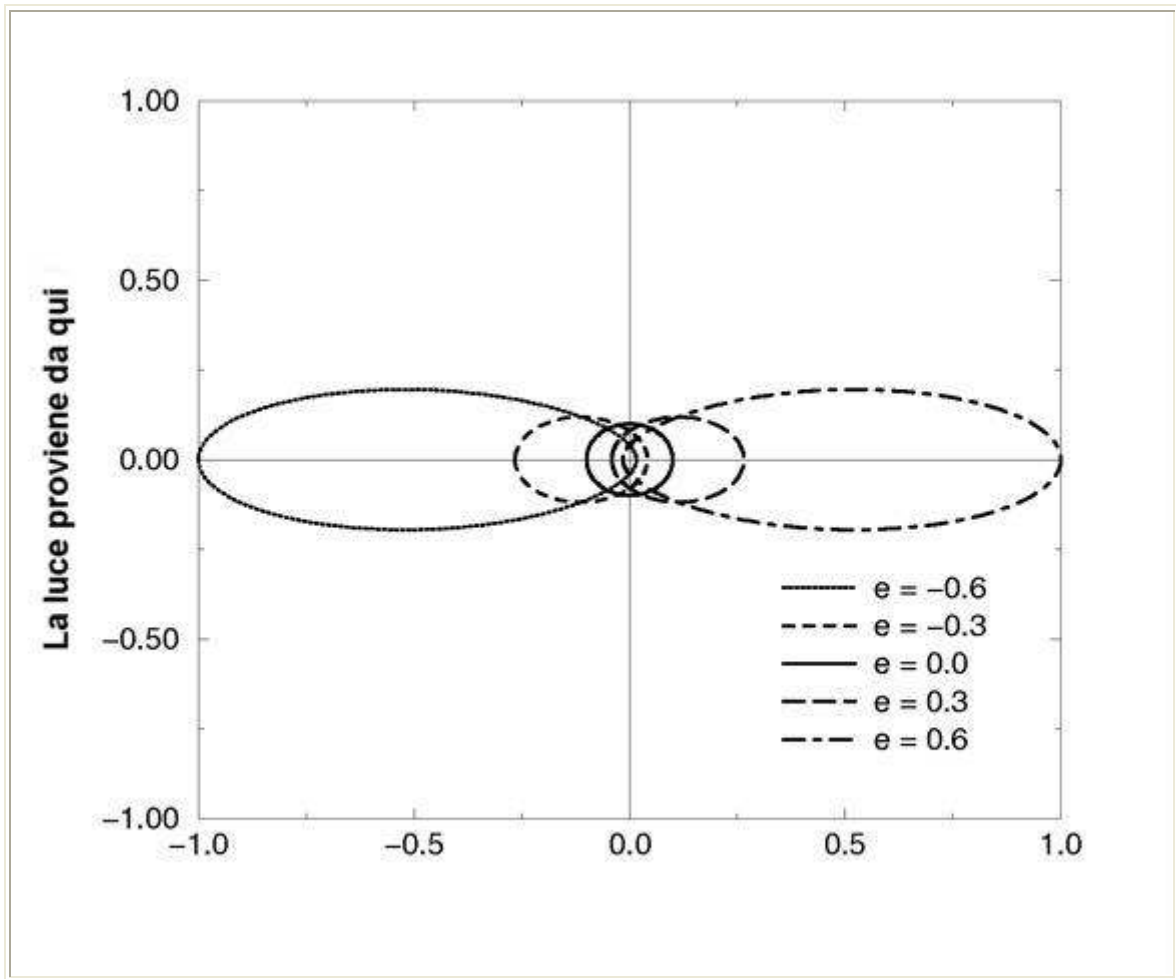



Figura 232- La funzione di dispersione di Henyey -Greenstein con diversi valori di eccentricità

Il modo più veloce per usare diversi tipi di dispersione sarà quello di dichiarare alcne costanti ed usarle nella definizione dell'atmosfera.

```
#declare ISOTROPIC_SCATTERING = 1
#declare MIE_HAZY_SCATTERING = 2
#declare MIE_MURKY_SCATTERING = 3
#declare RAYLEIGH_SCATTERING = 4
#declare HENYEY_GREENSTEIN_SCATTERING = 5
```

la parola chiave `distance` è usata per determinare la densità delle particelle nell'atmosfera. Questa densità è costante nell'intera atmosfera e il parametro `distance` funzionerà allo stesso modo visto per la nebbia.
Con la parola chiave `scattering` è possibile modificare la quantità di luce che è dispersa dall'atmosfera aumentandone o diminuendone la luminosità. Valori più piccoli di dispersione diminuiranno la luminosità mentre valori più alti l'aumenteranno.
La parola chiave `color` o `colour` può essere usata per creare atmosfere colorate, cioè può essere usata per creare particelle che filtreranno la luce che le attraversa. Il colore predefinito è nero.
La luce che passa attraverso l'atmosfera (sia proveniente dalle sorgenti luminose, sia dallo sfondo) è filtrata dal colore dell'atmosfera se il colore specificato ha un valore diverso da zero. In altre parole, la quantità di luce che è filtrata dall'atmosfera è data dal valore che è stato assegnato al filtro (nello stesso modo in cui si comporta la nebbia). Usando un colore `rgbf <1, 0, 0, 0.25>` si

otterrà un'atmosfera rossastra perché il 25% della luce che passa attraverso l'atmosfera sarà filtrato dal colore dell'atmosfera, cioè `rgb <1, 0, 0>` (rosso).
Il canale di trasmittanza del colore dell'atmosfera è utilizzato per specificare una quantità minima di trasparenza. Se viene usato un valore maggiore di zero sarà sempre possibile vedere una percentuale di sfondo relativa al valore assegnato attraverso l'atmosfera, come per la nebbia.
Poiché l'atmosfera è calcolata tramite un campionamento lungo la visuale dell'osservatore e tenendo conto delle sorgenti luminose, è soggetta ad aliasing (come ogni altra tecnica di campionamento). Ci sono quattro parametri per minimizzare i difetti che si possono verificare : `samples`, `jitter`, `aa_level` e `aa_threshold`.
La parola chiave `samples` determina quanti campioni sono calcolati in un intervallo lungo il raggio visivo. La lunghezza dell'intervallo è, o la distanza specificata dal valore del parametro `distance` o la lunghezza della parte illuminata del raggio che è sempre minore. Questa parte del raggio è la sezione che è *più probabilmente* colpita dalla luce. Nel caso di uno spot è la parte del raggio che giace nel cono di luce. Negli atri casi, la sua determinazione diventa più difficile. L'unica cosa da ricordare è che la lunghezza dell'intervallo di campionamento è variabile, ma non sarà mai minore dei campioni specificati nella distanza specificata.
Una tecnica per ridurre gli errori dovuti al campionamento è di applicare il jittering ai punti campionati. In altre parole, di aggiungere un'interferenza casuale alla loro posizione. Si può fare questo usando la parola chiave `jitter`.
Un'altra tecnica è il sovracampionamento (il metodo anti-aliasing) : questo aiuta a eliminare i difetti dovuti al campionamento aggiungendo campioni supplementari dove avvengono cambiamenti di grande entità (ad esempio, ai contorni di un'ombra). L'anti-aliasing viene attivato dalla parola chiave `aa_level`. Se il valore assegnatole è maggiore di zero verrà usato il sovracampionamento. I campioni aggiuntivi saranno posizionati ricorsivamente tra due campioni che presentano valori molto diversi. Il livello a cui cessa quest'iterazione viene specificato dalla parola chiave `aa_level`. Un valore di 1 significa una suddivisione (un campione supplementare), 2 significa due suddivisioni (fino a tre campioni supplementari) ecc.
la soglia per la differenza tra due campioni adiacenti è data dalla parola chiave `aa_threshold`. Se la differenza di intensità è maggiore di questa soglia verrà usato l'anti-aliasing per i due campioni.
Con gli spot si potranno ottenere i migliori risultati poiché il loro cono di luce diventerà visibile. Le luci puntiformi possono essere usate per creare effetti come lampioni nella nebbia. Si può evitare che le luci interagiscano con l'atmosfera aggiungendo il comando `atmosphere off` alla sorgente luminosa. Le luci che non interagiscono con la scena possono essere usate per aumentare il livello di luce complessivo per rendere il risultato maggiormente realistico.
L'atmosfera non funzionerà se la macchina fotografica si trova all'interno di un oggetto non vuoto, vedi il paragrafo "Oggetti Vuoti e Oggetti Solidi".

7.7.2 Sfondo

E' possibile specificare un colore per lo sfondo. Tutti i raggi che non colpiranno oggetti verranno colorati in quel modo. Il colore predefinito dello sfondo è nero. La sintassi è :

```
background { colour <COLORE> }
```

7.7.3 Nebbia

La nebbia è definita dalle seguenti frasi :

```
fog {
fog_type TIPO_DI_NEBBIA
distance DISTANZA
```

```
colour <COLORE>
[ turbulence <TURBOLENZA> ]
[ turb_depth PROFONDITA'_DELLA_TURBOLENZA ]
[ omega OMEGA ]
[ lambda LAMBDA ]
[ octaves OTTAVE ]
[ fog_offset FOG_OFFSET ]
[ fog_alt FOG_ALT ]
[ up <FOG_UP> ]
[ TRASFORMAZIONI ]
}
```

Il vettore `up` facoltativo, definisce l' "alto" e generalmente si usa lo stesso che si è usato per la camera. Tutti i calcoli fatti per la nebbia raso terra saranno dipendenti da questo vettore, in altre parole le altezze relative saranno calcolate lungo questo vettore.

Questo vettore può essere modificato usando tutte le trasformazioni descritte nel paragrafo "Trasformazioni". Anche se potrebbe non essere una buona idea scalare il vettore `up` (è assai difficile riuscire a prevedere il risultato), è piuttosto utile al contrario saperlo ruotare. Le traslazioni invece non lo influenzano e quindi non muteranno neanche la posizione della nebbia.
Allo stato attuale sono disponibili due tipi di nebbia : la *nebbia costante* e la *nebbia raso terra*. La nebbia costante ha una densità costante ovunque, a tutte le altezze minori di un determinato punto che giace sull'asse dell'altezza. L'altezza sotto alla quale la nebbia risulterà costante è specificata dalla parola chiave `fog_offset`. La parola chiave `fog_alt` specifica la velocità alla quale la nebbia si dilegua. All'altezza `fog_offset+fog_alt` la nebbia ha una densità del 25%. La densità della nebbia ad una determinata altezza è data dalla seguente

$$densità = \begin{cases} \dfrac{1}{\left[1 + \left(y - fog_offset\right) / fog_alt\right]^2}, & y > fog_alt \\ 1, & y \leq fog_alt \end{cases}$$

formula :

La densità totale lungo un raggio è calcolata dall'integrale che varia tra l'altezza del punto iniziale e l'altezza del punto finale.
Nel file **const.ini** sono definite due costanti riguardanti la nebbia :

```
// FOG TYPE CONSTANTS
#declare Constant_Fog = 1
#declare Ground_Fog = 2
```

Il colore di un pixel con una certa profondità di intersezione è calcolato con la seguente formula :

```
C_pixel = exp(-d/D) * C_oggetto + (1-exp(-d/D)) * C_nebbia
```

dove D è la distanza della nebbia. A profondità zero il colore risultante sarà quello dell'oggetto. Se la profondità dell'intersezione uguaglia la distanza della nebbia il colore finale sarà costituito per il 64% dal colore dell'oggetto e per il 36% dal colore della nebbia.
Il colore della nebbia impostato tramite la parola chiave `color` ha tre funzioni. Primo, definisce il colore da usare nel passaggio dalla nebbia allo sfondo. Secondo, viene usato per specificare la soglia di trasparenza. Usando una trasparenza maggiore di 0, si può infatti essere sicuri che al limite sarà possibile vedere quella quantità di luce attraverso la nebbia. Con una trasparenza di 0.3 si vedrà

come minimo il 30% dello sfondo. Terzo, può essere usata per creare una nebbia filtrante. Con una quantità di filtro maggiore di zero la luce dello sfondo sarà determinata moltiplicando il valore del filtro per il colore dello sfondo. Un filtro di 0.7 produrrà una nebbia che filtrerà il 70% dello sfondo.
La nebbia può essere stratificata. Si possono usare tanti strati quanti se ne desiderano. Generalmente l'effetto migliore si ha quando ogni strato è uno strato di nebbia raso terra di un colore diverso, con altezza e turbolenza diversa. Per usare molti strati di nebbia, basta aggiungerli alla scena.
È possibile anche creare una nebbia più "mossa" aggiungendole della turbolenza. La parola chiave `turbulence` deve essere seguita da un decimale o da un vettore per specificare la quantità di turbolenza da usare. Si possono specificare anche i parametri omega, lambda e le ottave. Vedere il paragrafo "Modificatori di Pattern" per maggiori informazioni su questi parametri relativi alla turbolenza.
In più la turbolenza della nebbia può essere scalata lungo la direzione del raggio visivo usando la parola chiave `turb_depth`. Valori ottimali possono variare tra 0.0 e 1.0. Il valore predefinito è 0.5 ma è possibile usare qualunque valore decimale.
La nebbia non funzionerà se la camera è dentro ad un oggetto non vuoto. (Vedere il paragrafo "Oggetti Vuoti ed Oggetti Solidi" per spiegazioni più approfondite).

7.7.4 Sfera Celeste

La sfera celeste è utilizzata per creare cieli realistici da usare come sfondo senza bisogno di sfere aggiuntive per simulare il cielo. La sintassi è :

```
sky_sphere {
pigment { PIGMENTO1 }
pigment { PIGMENTO2 }
pigment { PIGMENTO3 }
...
[ TRASFORMAZIONI ]
}
```

La sfera celeste può contenere diversi strati di pigmenti, con l'ultimo pigmento in cima, in pratica quello calcolato per ultimo e il primo in fondo, vale a dire quello calcolato per primo. Se gli strati superiori conterranno filtri o trasparenze gli strati inferiori risplenderanno attraverso essi, se invece gli strati superiori non conterranno filtri o trasparenze gli strati inferiori non si vedranno.
La sfera celeste è calcolata usando il vettore direzione come riferimento per i motivi di colore. Questo metodo assicura comunque un risultato che modellerà un bel cielo, che si troverà a distanze molto grandi rispetto alle posizioni degli oggetti indipendentemente dall'angolo di visuale.
Se vuoi aggiungere una bella sfumatura di colore al tuo sfondo, usa il prossimo esempio.

```
sky_sphere {
pigment {
gradient y
color_map {
[ 0.5 color CornflowerBlue ]
[ 1.0 color MidnightBlue ]
}
scale 2
translate -1
}
}
```

Questo da una sfumatura `CornflowerBlue` all'orizzonte per arrivare a `MidnightBlue` allo zenit. Le operazioni di ridimensionamento e traslazione sono usate per mappare i valori del vettore

direzione, che varia dal valore <-1, -1, -1> al valore <1, 1, 1> e farlo variare tra <0, 0, 0> e <1, 1, 1> . Così viene evitata la ripetizione della sfumatura di colore per le parti che si trovano sotto l'orizzonte.
Per movimentare facilmente la sfera celeste è possibile utilizzare le solite operazioni di trasformazione descritte nel paragrafo "Trasformazioni".
E' possibile usare solo una sfera celeste per volta e non tutto funzionerà correttamente anche se si utilizzeranno la camera ortografica o cilindrica. La camera ortografica usa raggi paralleli e quindi permette di vedere una parte molto piccola della sfera celeste (nella maggior parte dei casi si otterrà un cielo di un unico colore). Funzionano le riflessioni sulle superfici curve, cioè si potranno chiaramente vedere i riflessi specchiati del cielo su una sfera.

7.7.5 Arcobaleno

E' possibile generare arcobaleni usando archi circolari, simili alla nebbia. La sintassi completa è :

```
rainbow {
direction <DIREZIONE>
angle ANGOLO
width LUNGHEZZA
distance DISTANZA
color_map { MAPPA_DEI_COLORI }
[ jitter JITTER ]
[ up <UP> ]
[ arc_angle ANGOLO_DELL'_ARCO ]
[ falloff_angle ANGOLO_DI_DECADIMENTO]
}
```

Il vettore `direction` determina la direzione della luce (virtuale) che causa l'arcobaleno. Idealmente questa è una sorgente luminosa infinitamente lontana che emette raggi di luce paralleli (come il sole). La posizione e la dimensione dell'arcobaleno sono specificate dalle parole chiave `angle` e `width`. Per capire come funzionano si dovrebbe prima sapere come viene calcolato l'arcobaleno.
Per ogni raggio si calcola l'angolo tra il vettore direzione dell'arcobaleno e il vettore direzione del raggio di luce. Se questo angolo si trova nell'intervallo che va da `ANGLE-WIDTH/2` a `ANGLE+WIDTH/2` l'arcobaleno è colpito dal raggio. Il colore viene quindi determinato utilizzando l'angolo come indice nella mappa di colori dell'arcobaleno. Dopo che il colore è stato determinato verrà unito al colore dello sfondo come avviene per la nebbia. In questo modo i parametri `angle` e `width` determinano l'angolo sotto il quale l'arcobaleno è visibile. La parola chiave `jitter` può essere usata per aggiungere perturbazioni casuali alla mappa dei colori. Ciò aggiunge qualche irregolarità all'arcobaleno e lo rende più realistico.
La parola chiave `distance` è la stessa che viene usata per la nebbia. Dato che l'arcobaleno è un effetto simile alla nebbia è possibile che esso sia visibile sugli oggetti. Se questo effetto è indesiderato può essere evitato utilizzando un alto valore per la distanza. Per default si usa un valore abbastanza alto da far sì che questo effetto non si presenti.
La parola chiave `color_map` è usata per definire una mappatura del colore che verrà distribuita sull'arcobaleno. Per creare arcobaleni realistici è importante sapere che l'indice della mappa aumenta con l'angolo trai vettori direzione del raggio e dell'arcobaleno. L'indice è 0 sull'anello più interno e 1 sull'anello più esterno. I valori di filtro e di trasmittanza nei colori della mappa hanno lo stesso significato di quelli usati coi vari tipi di nebbia (vedi paragrafo "Nebbia").
L'arcobaleno predefinito è un cerchio di 360°. Questo non è un problema fino a quando si ha un piano che nasconde la parte inferiore dell'arcobaleno. Se non vuoi che si veda tutto l'arco puoi usare le parole chiave opzionali `up`, `arc_angle` e `falloff_angle` per specificare un arco più

piccolo.
La parola chiave `arc_angle` specifica la misura dell'arco in gradi, da zero a 360. Un valore minore di 360° dà un arco con le estremità molto nette. Dal momento che questo non ha un bell'aspetto puoi usare la parola chiave `falloff_angle` per specificare una regione nella quale l'arcobaleno sfumerà lentamente nel colore dello sfondo. L'angolo di falloff deve essere minore o uguale all'angolo dell'arco.
La parola chiave `up` determina dove è posizionato l'angolo zero. Cambiando questo vettore puoi ruotare l'arcobaleno nella direzione che esso indica. Puoi notare che l'arco parte da `-arc_angle/2` e termina a `+arc_angle/2`. La regione sfumata va da `-arc_angle/2` a `falloff_angle/2` e da `+falloff_angle/2` a `+arc_angle/2`.
Il seguente esempio crea un arcobaleno di 120° che ha un regione in cui viene sfumato ampia 30° ad entrambe le estremità.

```
rainbow {
direction <0, 0, 1>
angle 42.5
width 5
distance 1000
jitter 0.01
color_map { Rainbow_Color_Map }
up <0, 1, 0>
arc_angle 240
falloff_angle 60
}
```

E' possibile usare più di un arcobaleno e combinare ognuno degli arcobaleni con altri effetti atmosferici.

7.8 Impostazioni Globali

La frase `global_settings{...}` è una frase che comprende diversi parametri che riguardano la scena nel suo insieme. La frase può apparire ovunque all'interno della scena, dal momento che non è all'interno di nessun altro parametro. Si possono avere nello stesso file più di una frase di impostazioni globali. Comunque, i valori specificati nell'ultima frase di impostazioni sovrascrivono quelli precedenti. Indifferentemente dalla posizione in cui viene scritta la frase delle impostazioni globali, queste saranno comunque applicate all'intera scena.
Alcuni elementi che erano direttive del linguaggio nelle precedenti versioni di POV-Ray sono state incluse nelle specificazioni delle impostazioni globali. In questo modo è più esplicita l'assunzione che il loro effetto si applica a tutta la scena. La vecchia sintassi è ancora mantenuta ma provoca un messaggio di avvertimento.

```
global_settings {
adc_bailout DECIMALE
ambient_light COLORE
assumed_gamma DECIMALE
hf_gray_16 ON/OFF
irid_wavelength COLORE
max_intersections INTERO
max_trace_level INTERO
number_of_waves INTERO
radiosity { IMPOSTAZIONI_RADIOSITY... }
}
```

Ogni elemento è facoltativo e può apparire in qualunque ordine. Se un elemento è specificato più di una volta, l'ultima impostazione sovrascrive le precedenti. Dettagli su ogni elemento sono forniti nei prossimi paragrafi.

7.8.1 ADC_Bailout

Nelle scene con molte superfici riflettenti e trasparenti, il rendering può essere molto lento perché POV-Ray traccerà molti raggi per determinare le riflessioni e le rifrazioni, che però avranno un contributo piuttosto piccolo sul colore di un particolare punto. Il programma usa un sistema chiamato *adaptive depth control* (ADC) per limitare i raggi aggiuntivi quando il loro contributo non è particolarmente significativo. E' possibile usare la parola chiave `adc_bailout` nelle impostazioni generali per specificare il momento a partire dal quale il contributo di un raggio è considerato trascurabile.

```
global_settings { adc_bailout DECIMALE }
```

Il valore predefinito è di 1/255 (0.0039) poiché una differenza minore non sarebbe visibile in un immagine a 24 bit. Generalmente quest'impostazione è più che adeguata. Impostare `adc_bailout` a 0 disattiverà la funzione, lasciando a `max_trace_level` il compito di stabilire un limite superiore al numero dei raggi lanciati. Vedere il paragrafo "Max_Trace_Level" per dettagli su come i due parametri interagiscono.

7.8.2 Luce Ambiente

La luce ambiente è usata per simulare l'effetto della riflessione interdiffusa che è responsabile della tenue illuminazione delle aree che sono parzialmente o totalmente in ombra. POV-Ray prevede una luce ambiente per cambiare facilmente la luminosità della scena senza cambiare tutti i valori corrispondenti contenuti nelle frasi di `finish`. Permette anche di creare effetti interessanti cambiando il colore della luce ambiente.

```
global_settings { ambient_light COLORE }
```

Il valore predefinito è una luce ambiente bianca impostata a rgb <1, 1, 1>. Il valore effettivamente utilizzato è dato da :

```
LUCE_AMBIENTE=AMBIENTE_DEL_FINISH*AMBIENTE_DELLE_IMPOSTAZIONI
```

Vedi il paragrafo "Luce Ambiente" per maggiori dettagli.

7.8.3 Gamma Colore

Molte persone si possono essere accorte che talvolta alcune immagini sono troppo (o troppo poco) luminose quando vengono visualizzate sul loro sistema. In generale, gli utenti di Macintosh trovano che le immagini create su un PC sono troppo luminose, mentre coloro che usano i PC vedono le immagini create sui sistemi Macintosh come troppo scure.
La parola chiave `assumed_gamma` nelle impostazioni globali agisce in unione con l'impostazione del file .INI `Display_Gamma` (vedi il paragrafo "Impostazioni Hardware" ed assicura che la scena venga renderizzata nello stesso modo sulle varie piattaforme sulle quali può essere installato POV-Ray. L'impostazione `assumed_gamma` è usata nella scena aggiungendo la seguente frase :

```
global_settings { assumed_gamma DECIMALE }
```

dove il valore attribuito ad `assumed_gamma` è il fattore di correzione che deve essere applicato

prima che i punti dell'immagine vengano visualizzati o salvati su disco. Per le scene create con versioni precedenti di POV-Ray il valore di `assumed_gamma` sarà lo stesso che il valore di `Display_Gamma` del sistema su cui è stata creata la scena. Per i PC il valore più comune è 2.2, mentre per le scene create sui Macintosh sarà 1.8. Un altro valore comune è 1.0
Scene che non contengono nessuna specificazione per `assumed_gamma` non avranno nessuna correzione per ragioni di compatibilità. Se stai creando nuove scene, o renderizzando vecchie scene, è fortemente raccomandato l'uso dell'impostazione `assumed_gamma` appropriata. Per le nuove scene dovrai usare un valore di 1.0 poiché questo rende le scene molto più realistiche.
I seguenti paragrafi spiegano più approfonditamente che cos'è la *gamma colore* e perché è importante.

7.8.3.1 Gamma del Monitor

Le differenze nella visualizzazione dell'immagine sono un risultato dipendente dal modo in cui il computer elabora l'immagine e la visualizza sul monitor. Nel processo del rendering e della visualizzazione a schermo sono importanti alcuni valori di gamma, incluso il valore di gamma della scena o dell'immagine e quello del monitor. Molte immagini generate con POV-Ray immagazzinano numeri nell'intervallo 0-255 per ognuna delle componenti rossa, verde e blu del pixel. Questi numeri rappresentano l'intensità di ciascuna componente del colore, con 0 uguale al nero e 255 al colore puro (100% rosso, o verde, o blu). Quando un'immagine è visualizzata, la scheda grafica converte ciascuna componente di colore in un voltaggio che è mandato al monitor per accendere i fosfori rossi, verdi o blu dello schermo. Il voltaggio è solitamente proporzionale al valore di ciascuna componente del colore.
Il valore della gamma diventa importante quando si tratta di visualizzare intensità che costituiscono valori intermedi tra 0 e 255. Per esempio, 127 dovrebbe rappresentare il 50% del massimo dell'intensità per un punto. Sui sistemi che non applicano la correzione della gamma, 127 sarà convertito al 50% del massimo voltaggio, ma a causa del modo in cui funzionano i fosfori ed i cannoni elettronici di un monitor, questo valore può essere solo il 22% della massima intensità di colore su un monitor che ha un valore di gamma di 2.2. Per mostrare un punto che ha il 50% dell'intensità del colore su quel monitor, dovremmo usare un voltaggio pari al 73% del voltaggio massimo, che significa adottare un valore per quel punto di 186.
La relazione tra il valore di input del pixel e l'intensità della visualizzazione al monitor può essere approssimata con una funzione esponenziale :

```
oluce = iluce ^ display_gamma
```

dove `oluce` è l'intensità di output e `iluce` è l'intensità del pixel di input. Entrambi i valori stanno tra zero ed uno (dallo 0% al 100%). Molti monitor hanno un valore di gamma tra 1.8 e 2.6. Usare la formula vista ora con valori di gamma del monitor maggiori di 1 significherà che la luminosità dell'output sarà minore della luminosità dell'input. Volendo avere la luminosità dell'output e dell'input uguali si deve avere un valore di gamma per tutto il sistema uguale ad 1. Per ottenere ciò, dobbiamo correggere la luminosità dell'input nello stesso modo descritto sopra, ma con un valore di gamma di 1/gamma_del_monitor prima di inviarla al monitor. Per correggere una gamma del monitor di 2.2, questa correzione usa un valore di 1/2.2 (circa 0.45).

Il modo in cui la correzione di gamma pre-monitor viene effettuata dipende dall'hardware e dal software che si stanno usando. Sui sistemi Macintosh, il sistema operativo si occupa di uniformare le differenze tra le applicazioni e l'hardware. Attraverso un pannello di controllo della gamma, l'utente può impostare il valore di gamma del monitor e MacOS converte tutte le intensità dei pixel in modo che il monitor apparirà come se avesse uno specifico valore di gamma. Sulle macchine Silicon Graphics, la scheda video ha la correzione della gamma incorporata e calibrata al monitor, dando così il valore di gamma complessivo desiderato (il valore di default è 1.7). Sfortunatamente,

sui PC e sulla maggior parte dei sistemi UNIX dipende dall'applicazione effettuare le correzioni di gamma necessarie.

7.8.3.2 Gamma del File Immagine

Dato che la maggior parte delle applicazioni e dei formati delle immagini su Personal Computer e macchine Unix non implementano la gamma colore del monitor, non fanno niente per correggerla. Ne risulta che quando un utente di queste macchine crea un immagine, la modifica in modo da ottenere una luminosità adatta al monitor sul quale la osserva. Quindi, i dati memorizzati nel file immagine sono *già* corretti del fattore 0.45 che abbiamo visto prima. Quando questi file sono visualizzati su sistemi Macintosh, la correzione della gamma già presente nel file, sommata a quella operata automaticamente da MacOS, rende l'immagine troppo luminosa. Analogamente, le immagini che vengono visualizzate correttamente su sistemi Macintosh o SGI a causa della correzione della gamma automatica, saranno troppo scure se visualizzate su un PC.
I nuovi file in formato PNG generati da POV-Ray 3.0 superano questo problema memorizzando le informazioni sulla gamma dell'immagine (che è il reciproco della gamma del monitor). Quando il file in formato PNG viene visualizzato da un programma impostato correttamente, utilizza il valore di gamma incluso nel file insieme al valore di gamma del monitor per correggere le differenze tra i sistemi.
Sfortunatamente, di tutti i formati supportati da POV-Ray, PNG è l'unico che ha funzioni incorporate di correzione della gamma cromatica e quindi viene preferito per immagini che saranno visualizzate su un'ampia varietà di piattaforme.

7.8.3.3 Gamma del File Scena

Il problema della gamma colore del file immagine è il risultato del metodo con cui le immagini vengono generate in POV-Ray. Quando inizi una nuova scena e posizioni le luci e imposti le texture ed i colori delle superfici, in generale fai molti tentativi fino a quando l'illuminazione complessiva della scena non ti soddisfa. Il modo in cui fai queste scelte è guidato dall'immagine di anteprima, o dal file immagine memorizzato su disco, che a loro volta sono dipendenti dalla gamma dell'hardware video (scheda e monitor) che usi.

Ciò significa che tu stesso stai effettuando delle correzioni della gamma nella scena di POV-Ray in relazione al tuo hardware. La scena genera un'immagine che a sua volta ha una correzione della gamma appropriata all'hardware e che, di conseguenza, viene visualizzata correttamente su sistemi simili al tuo. Quando questa scena viene renderizzata su di un'altra piattaforma, può risultare troppo luminosa o troppo scura, indipendentemente dal formato utilizzato per il file immagine. Invece di modificare tutte le scene per ottenere un unico valore di gamma, POV-Ray 3.0 permette di specificare nel file della scena il valore della gamma colore del sistema su cui l'immagine è stata creata.
L'impostazione `assumed_gamma`, in associazione con il parametro .INI `Display_Gamma` istruisce POV-Ray su come correggere la gamma colore di una determinata scena in modo che sia l'immagine di anteprima che il file immagine di output abbiano la corretta luminosità su tutti i sistemi. Dato che la correzione della gamma è eseguita da POV-Ray, produce immagini della luminosità giusta per l'hardware video usato, indipendentemente dal formato del file di output utilizzato. Inoltre la correzione si avvale del formato di dati ad alta precisione che POV-Ray utilizza internamente, per cui dà risultati migliori della correzione che si può effettuare dopo che il file è stato scritto su disco.
Per quanto non si notino differenze nell'output del proprio sistema con e senza l'impostazione `assumed_gamma`, questo parametro è importante qualora la scena debba essere renderizzata su di un'altra piattaforma.

7.8.4 HF_Gray_16

Questa impostazione è utile quando si utilizza POV-Ray per generare height field. La sintassi è :

```
global_settings { hf_gray_16 on/off }
```

dove on/off attiva o disattiva l'opzione. Se la parola chiave viene specificata senza i valori on od off, allora l'opzione viene attivata. Se non è specificato `hf_gray_16` in nessuna frase delle impostazioni globali, allora la si considera disattivata.
Quando la funzione è attivata, il file di output sarà nel formato proprio degli height field, con l'altezza di ogni punto dipendente dalla luminosità del pixel. La luminosità del pixel è calcolata nello stesso modo in cui le immagini a colori sono convertite in immagini a toni di grigio.

```
altezza = 0.3 * rosso + 0.59 * verde + 0.11 * blu
```

Attivare l'opzione `hf_gray_16` farà sì che l'anteprima sia visualizzata in bianco e nero anziché a colori. Ciò è per permetterti di vedere quale sarà l'aspetto dell'height field poiché alcuni formati di immagine immagazzinano le informazioni relative in un modo difficile da comprendere. Vedi il paragrafo "Height Field" per una descrizione di come gli height field di POV-Ray sono memorizzati a seconda del tipo di file.

7.8.5 Irid_Wavelenght

I calcoli dell'iridescenza dipendono dalle lunghezze d'onda dominanti dei colori primari : rosso verde e blu. I valori possono essere modificati utilizzando l'impostazione `irid_wavelenght` come segue :

```
global_settings { irid_wavelenght COLORE }
```

Il valore predefinito è `rgb <0.25,0.18,0.14>` ed ogni valore assegnato a filtro o trasmittanza viene ignorato. Questi valori sono proporzionali alle lunghezze d'onda (*wavelenght*) della luce solare, ma non rappresentano parametri reali.
In generale, i valori predefiniti dovrebbero essere adatti per tutte le situazioni, ma questa opzione viene fornita per poter sperimentare con altri valori.

7.8.6 Max_Trace_Level

In scene con molte superfici riflettenti e trasparenti, POV-Ray può perdersi a tracciare molte riflessioni e rifrazioni che contribuiscono molto poco al colore di un particolare pixel.
L'impostazione globale `max_trace_level` definisce un numero massimo di livelli ricorsivi per i quali POV-Ray calcola le riflessioni.

```
global_settings { max_trace_level INTERO}
```

Questa impostazione è usata quando un raggio si riflette o passa attraverso un oggetto trasparente e quando vengono proiettate le ombre. Quando un raggio colpisce una superficie riflettente lancia un secondo raggio per determinare cosa riflette quel punto. Questo è il primo livello di ricorsività. Se questo secondo raggio colpisce un'altra superficie riflettente viene lanciato un terzo raggio (secondo livello di ricorsività). Il livello massimo predefinito è 5. Una funzione che aumenta la velocità del rendering nella versione 3.0 è il *Controllo Adattivo di Profondità* (ADC, Adaptive Depth Control). Ogni volta che il programma lancia un nuovo raggio a causa di riflessione o rifrazione, il suo contributo al colore complessivo del pixel è ridotto dall'ammontare della riflessione o del filtro della superficie rifrangente. In certi casi questo contributo può essere irrilevante e non c'è motivo di tracciare altri raggi. L'ADC è la funzione che segue questo contributo e decide di non tracciare ulteriori raggi. Nelle scene che usano molte superfici riflettenti o rifrangenti questa funzione può

ridurre notevolmente il numero di raggi lanciati, rendendo sicuro l'utilizzo di valori molto più alti di `max_trace_level`. Questa diminuzione del contributo dato al colore è il risultato della moltiplicazione del colore per il valore della riflessione e/o del filtro di ogni superficie. Quindi, una superficie a specchio (`reflection 1`) o perfettamente trasparente (`filter 1`) non potrà essere ottimizzabile da ADC. I risultati del controllo effettuato da ADC possono essere visualizzati alle voci *Rays Saved* e *Highest Trace Level* sulla schermata di statistiche al termine del rendering.

7.8.7 Max_Intersections

POV-Ray utilizza normalmente 64 allocazioni di memoria interne per raccogliere i punti di intersezione tra raggi ed oggetti. Scene complesse possono causare un overflow in queste allocazioni. POV-Ray non si ferma, ma può renderizzare la scena in maniera non corretta. Quando il rendering termina, se compare la voce *I-Stack Overflows* nelle statistiche che vengono mostrate, è necessario aumentare il numero delle allocazioni utilizzando l'impostazione

```
global_settings {max_intersections INTERO}
```

Se la voce I-Stack Overflows rimane presente nelle statistiche, aumentare il numero degli stack fino a che non cessa.

7.8.8 Number_Of_Waves

I motivi `wave` e `ripples` sono generati sommando una serie di onde, ognuna leggermente diversa dalla altre per centro e dimensioni. Normalmente, vengono sommate dieci onde, ma questo numero può essere controllato modificando l'impostazione `number_of_waves` nel modo seguente :

```
global_settings {number_of_waves INTERO}
```

Modificando questo parametro si influenzano sia le `waves` che le `ripples` presenti in tutta la scena.

7.8.9 Radiosity

Nota Bene : la funzione radiosity è sperimentale per POV-Ray 3.0. C'è un'alta probabilità che questa funzione venga cambiata in futuro. Non possiamo garantire che le scene che usano questa funzione nella versione 3.0 appariranno identiche nelle versioni future o che rimanga la piena compatibilità con la sintassi attuale.

Il metodo *radiosity* è un algoritmo che calcola in modo più realistico la interriflessione diffusa della luce. Questo effetto può essere visualizzato immaginando di mettere una sedia bianca in una stanza con un tappeto blu, le pareti blu ed il soffitto blu. La sedia assumerà una sfumatura blu da ciò che la circonda. E' anche da notare che le aree in ombra che ti circondano non sono completamente nere anche se su di esse non brilla nessuna luce in modo diretto. La luce diffusa che è riflessa dagli altri oggetti schiarisce le ombre. Normalmente il raytracing usa un metodo di simulazione di questo effetto chiamato *luce ambiente*, ma non è molto accurato.
Il metodo radiosity è molto più accurato della semplice luce ambiente, ma richiede molto più tempo di calcolo. Per questo motivo, non è normalmente utilizzato da POV-Ray. Per utilizzarlo è necessario utilizzare il parametro `+qr` nella linea di comando, o l'opzione .INI `Radiosity`. I Paragrafi seguenti spiegano come funziona il metodo radiosity, come controllarlo mediante impostazioni globali e forniscono alcuni suggerimenti sul rapporto qualità/velocità dei rendering che fanno uso di questa funzione.

7.8.9.1 Come Funziona la Radiosity

Il problema principale del raytracing è di scoprire quale è il livello di illuminazione di ogni punto visibile in una scena. Normalmente, l'illuminazione di un punto è divisa in questi quattro componenti :

Luce diffusa : l'effetto che rende più chiari i lati degli oggetti che sono di fronte alla luce ;
Luce speculare : l'effetto che fa sì che gli oggetti lucidi abbiano dei riflessi più chiari ;
Riflessione : l'effetto 'specchio'
Luce Ambiente : il livello complessivo di luce che ha la scena e che impedisce alle zone in ombra di essere totalmente oscurate.

Il sistema di calcolo radiosity di POV-Ray, basato su un metodo di Greg Ward, fornisce la via per sostituire l'ultimo di questi termini, la luce ambiente costante con un'illuminazione basata su *quali* superfici sono vicine ad un oggetto e quanto sono illuminate.
La prima cosa che si dovrebbe notare riguardo a questa definizione è che essa è circolare : l'illuminazione di ogni oggetto dipende da tutti gli altri e viceversa. Ciò è vero nel mondo reale, ma nel raytracing possiamo (e dobbiamo) accontentarci di un'approssimazione. L'approssimazione che viene usata è : gli oggetti che osservi hanno il loro valore di luce ambiente che viene calcolato sulla base dell'ambiente circostante, ma durante i calcoli viene usata l'illuminazione ambiente costante.
Come fa POV-Ray a calcolare il termine corrispondente alla luce ambiente per ogni punto ?
Lanciando raggi supplementari, in direzioni diverse e facendo una media dei risultati. Un punto normalmente può utilizzare 200 o più raggi per ritornare un valore corretto di luce ambiente. Questo fa pensare che il rendering sarebbe 200 volte più lento ed è vero, tranne che per il fatto che il software si avvantaggia del cambiamento molto graduale dei valori della luce ambiente (le ombre vengono calcolate separatamente, per cui i loro bordi, anche se sono molto netti, non costituiscono un problema). In questo modo questi raggi supplementari sono lanciati *solo una volta* (ogni 50 pixel circa) e poi i valori che ne risultano vengono memorizzati e riutilizzati per i punti vicini dell'immagine, quando possibile).
Il processo di memorizzare e riutilizzare i valori provoca la necessità di un certo numero di parametri che lo regolino, in modo da potere ottenere la scena che desideri.

7.8.9.2 Regolare la Radiosity

Come abbiamo detto prima, il metodo radiosity viene attivato utilizzando l'opzione .INI `Radiosity` o il parametro `+qr` nella linea di comando. La radiosity ha comunque molti parametri che vengono specificati nella frase `radiosity{...}` che si trova all'interno delle impostazioni globali, come segue.

```
global_settings {
radiosity {
brightness DECIMALE
count INTERO
distance_maximum DECIMALE
error_bound DECIMALE
gray_threshold DECIMALE
low_error_factor DECIMALE
minimum_reuse DECIMALE
nearest_count INTERO
recursion_limit INTERO
}
}
```

Ogni linea è opzionale e può apparire in qualunque ordine. Se una linea è specificata più di una

volta, l'ultimo dei valori specificati sovrascrive gli altri. Dettagli su ogni parametro sono forniti nei capitoli seguenti.

7.8.9.2.1 Brightness (luminosità)

Questo parametro rappresenta il grado luminosità da aggiungere al valore di luce ambiente. Se un oggetto è rosso `rgb<1,0,0>`, con un valore per il parametro `ambient` di 0.3, in una situazione normale verrà aggiunta una componente di rosso di 0.3. Se è attivata la radiosità viene considerato come circondato da un oggetto di colore <0.6,0.6,0.6>. Il colore medio che deriverà dopo i calcoli sarà immutato. Questo colore sarà moltiplicato per il valore assegnato al parametro `ambient` della texture, pari a 0.3,dando come risultato il colore <0.18,0.18,0.18>. questo colore è molto più scuro di 0.3 e sarà aggiunto al colore definito. Tutti i valori risultanti saranno schiariti dell'inverso della media dei valori calcolati, cosicché non cambi il valore medio di `ambient` aggiunto. Qualche volta sarà maggiore di quello specificato (maggiore di 0.3 nell'esempio) e qualche volta minore, ma la luminosità dell'intera scena non cambierà.
Il valore impostato è 3.3

7.8.9.2.2 count

Il numero di raggi che sono lanciati quando viene definito un nuovo valore per la radiosità è definito dal parametro `count`. Sono appropriati valori intorno a 100 o 150. Può essere necessario usare valori maggiori per le scene che abbiano contrasti molto elevati tra i livelli di luce o con piccole zone nettamente illuminate. È preferibile usare questo parametro solo per il rendering finale poiché richiede un calcolo piuttosto lungo e laborioso. Dal momento che molte scene considerano il valore di `ambient` impostato dall'1% al 2% dei pixel, i vostri rendering si allungheranno dall'1% al 2% di questo numero in più rispetto al tempo precedente. Se si imposta il valore di `count` a 300 il rendering impiegherà da 3 a 6 volte di più (da 1% a 2% di 300).
Quando questo valore è troppo basso, il livello delle luci sembrerà un po' irregolare come se la superficie fosse leggermente increspata. Se questo effetto non è importante e non sciupa la tua scena (come, ad esempio, quando si è usata una mappatura delle normali) allora è meglio utilizzare un valore più basso.
Il valore predefinito è 100.

7.8.9.2.3 distance_maximum

Il parametro `distance_maximum` è l'unico il cui valore dipenda dalla dimensione degli oggetti nella scena. Questo parametro è l'unico che **deve** essere usato per renderizzare le scene con accuratezza, gli altri si possono tralasciare per i primi tentativi. È difficile descriverne il significato in modo semplice : imposta la distanza nelle unità di grandezza del modello in modo che l'errore tocchi il 100% `(radiosity_error_bound >=1)`: i campioni che si trovano a distanze superiori a questa dal punto iniziale non vengono riutilizzati.
Immagina una mela sul bordo sinistro di un tavolo. Lo scopo è quello di assicurarsi che i campioni alla destra del tavolo non siano troppo vicini alla mela e comunque non sotto la mela. Se ci sono abbastanza raggi questo non è un problema poiché uno di essi colpirebbe certamente la mela ed imposterebbe il raggio di riutilizzo dei campioni in maniera appropriata. In pratica, devi limitare questo fenomeno.
Viene usata questa tecnica : si trova l'oggetto nella scena che può presentare le seguenti difficoltà : essere un oggetto piccolo su una superficie piana molto grande e che deve essere circondato da una buona luce ambiente. Ora, quanto lontano devi essere dall'oggetto per avere una buona probabilità che l'oggetto sia colpito da un raggio ? Nell'esempio con la mela sul tavolo, considerando che si stia usando una unità di POV-Ray uguale ad un pollice, dovresti usare una distanza di 30 pollici. Teoricamente (quando si utilizza un alto numero di raggi) questa distanza è quella alla quale la cima dell'oggetto si trova a 5° sull'orizzonte del punto campione che si sta usando. Questo corrisponde a

circa 11 volte l'altezza dell'oggetto. Così, per una mela alta 3 pollici, potrebbe avere senso utilizzare una distanza di 33 pollici. Per rendere bene anche lo spazio intorno alla mela, usate 3 pollici ecc. un altro modo approssimativo per calcolarla è utilizzare 1/3 della distanza tra il punto di osservazione e il punto che si sta guardando. E ragionevole che tu non sia a più di 90 pollici dalla mela sul tavolo, se ti interessa vedere l'ombreggiatura attorno all'oggetto. Il valore predefinito è 0.

7.8.9.2.4 error_bound

Il parametro `error_bound` è uno dei due principali fattori che regolano il rapporto velocità/qualità (l'altro è ovviamente il numero di raggi usati). In un mondo ideale, sarebbe il *solo* valore necessario. Esso significa, semplicemente, la frazione di errore tollerata. Per esempio, se fosse impostato ad 1, l'algoritmo non calcolerebbe un nuovo valore fino a che l'errore sul precedente non avesse raggiunto il 100%. Ignorando per il momento l'errore introdotto dalla rotazione, sulle superfici piatte, esso è uguale ad una frazione della distanza di riutilizzo, che a sua volta è la distanza dall'oggetto colpito più vicino. Se hai un precedente campione sul pavimento a 10 pollici dal muro, un valore di `error_bound` di 0.5 genererebbe un nuovo campione a 5 pollici dal muro. Il valore 0.5 è abbastanza sbrigativo, 0.33 è valido per rendering finali. Valori molto inferiori a 0.33 causano dei rendering **eterni**.
Il valore predefinito è 0.4

7.8.9.2.5 gray_threshold

La luce causata dalla riflessione interdiffusa in un punto è una funzione degli oggetti che si trovano attorno ad esso. Dato che questa funzione è definita ricorsivamente fino a milioni di livelli di ricursione, in ogni scena reale, ogni oggetto è illuminato almeno in parte da ogni altro oggetto della scena. Poiché non possiamo permetterci di calcolare questo effetto, facciamo un passaggio solo e quindi la luce ambiente calcolata per un oggetto è molto influenzata dai colori degli oggetti vicini. Questo effetto avviene realmente, ma non così tanto come questo metodo di calcolo potrebbe far credere. La variabile `gray_threshold` ingrigisce un poco l'effetto, per rendere la scena più credibile. Un valore di 0.6 significa che la luce ambiente risultante viene composta per il 60% dei toni di grigio equivalenti ai colori calcolati e per il restante 40% dai colori effettivamente calcolati. Impostata a 0, questa funzione non ha effetto, mentre al 100% si ottengono luci ambiente bianche e grigie, senza tonalità di colore. E' da notare che non viene modificata la luminosità della luce ambiente, ma solo la saturazione del colore.
Il valore predefinito è 0.5.

7.8.9.2.6 low_error_factor

Se si calcola solo il numero necessario di campioni, ma non di più, si otterrà un'immagine dall'illuminazione leggermente 'chiazzata'. Ciò che si desidera è che sia leggermente più diffusa, in modo che le sfumature siano graduali e di bell'aspetto. La soluzione a questo problema è l'anteprima a mosaico : prima passa una o più volte l'immagine calcolando i valori di radiosity. Per assicurarsi di averne un numero leggermente superiore al necessario, l'algoritmo di radiosity abbassa il valore di `error_bound` durante i passaggi preliminari, per poi riportarlo al valore corretto subito prima del rendering vero e proprio. Il parametro `low_error_factor` imposta un fattore per il quale `error_bound` viene moltiplicato durante i passaggi preliminari. Ad esempio, se `error_bound` è 0.4 e `low_error_factor` è 0.8, il programma userà un valore effettivo di 0.32 per i passaggi preliminari e di 0.4 per il rendering finale.
Il valore predefinito è 0.8.

7.8.9.2.7 minimum_reuse

Il raggio minimo effettivo è impostato mediante `minimum_reuse`. Questa è la frazione della larghezza dello schermo che imposta il raggio minimo di riutilizzo per ogni campione (in effetti, è

frazione della distanza dall'osservatore, ma le due sono circa uguali). Ad esempio, se il valore è 0.02 il raggio di riutilizzo massimo è impostato al 2% della larghezza dello schermo. Immagina di mandare un raggio all'orizzonte e che esso colpisca il suolo ad una distanza di 100 miglia da te. La distanza di riutilizzo per quel campione sarebbe di due miglia. Ad una risoluzione di 300x400 punti, ciò corrisponderebbe circa ad 8 pixel. Diciamo che non vuoi stare a calcolare i valori per ogni pixel dentro ad ogni crepa della scena, poiché ci vorrebbe troppo. Il parametro `minimum_reuse` imposta un limite minimo per il riutilizzo dei campioni. Se questo valore è troppo basso, il tempo di rendering aumenta e gli angoli interni risultano leggermente sgranati. Se è troppo alto, non otterrai la naturale oscurità degli angoli nascosti della scena, dato che il programma riutilizza i campioni anche per quelli. A valori maggiori del 2% (0.02) si cominciano ad avere errori più palesi, come riutilizzare la luce del tavolo sotto la mela.
E' da ricordare che questo valore è un numero puro. Il valore predefinito è di 0.015.

7.8.9.2.8 nearest_count

Il valore di `nearest_count` è il numero massimo dei vecchi valori della luce ambiente che possono essere combinati per ottenerne uno nuovo mediante interpolazione. Ad essere riutilizzati saranno sempre gli n punti geometricamente più vicini. Se il valore assegnato a `nearest_count` diventa più piccolo di 4, i risultati possono diventare piuttosto irregolari, ma può essere buono per la correzione della scena. Questo valore non deve comunque superare 10, che è la dimensione del vettore allocato. Il valore predefinito è 6.

7.8.9.2.9 radiosity_quality

Questa funzione non è ancora stata implementata.

7.8.9.2.10 recursion_limit

Questo valore determina quanti livelli di ricursione devono essere usati per calcolare la radiosity. I valori validi sono 1 e 2. Il valore predefinito è 1.

7.8.9.3 Suggerimenti sulla Radiosity

Se vuoi vedere dove sono calcolati i valori di radiosity, imposta a 20 `radiosity_count`, ad 1 `radiosity_nearest_count` e `radiosity_gray` a 0. Ciò renderà le zone in cui viene calcolata la radiosity simili a chiazze di colore. Bene, i calcoli di radiosity vengono eseguiti al centro della maggior parte delle chiazze. In più, questa prova è veloce da eseguire. Inoltre, per vedere come l'immagine ne risulta influenzata, pui provare a cambiare i valori di `radiosity_error_bound`, di `radiosity_reuse_dist_min` e di `radiosity_reuse_dist_max`.
Un modo per ottenere risultati estremamente sfumati : aumentare il numero di campioni (siamo arrivati a 1300) e ridurre `low_error_factor` a qualcosa di piccolo come 0.6. Aumentare `reuse_count` a 7 od 8. Ciò fornirà valori migliori ed in numero superiore e poi li interpolerà con altri ancora nell'ultimo passaggio. Non è adatto a persone con poca pazienza dato che il tempo di rendering aumenta in maniera geometrica. Se hai delle irregolarità nella luce ambiente solo in certe zone, o vicino a certi oggetti, prova invece a modificare `error_bound`. Non abbassarlo mai più di un poco per volta perché il tempo di rendering aumenterà molto.
Se la scena che ne risulta è corretta, ma ottieni, vicino ad alcuni oggetti, punti del colore giusto (di solito il più scuro) su superfici del colore sbagliato, prova ad abbassare `reuse_dist_max`. Se non funziona ancora, aumenta a 100 il numero di raggi ed abbassa molto poco `error_bound`. Se ci sono ancora problemi, abbassa `reuse_nearest_count` a circa 4.

Appendice A Copyright

I seguenti paragrafi contengono le informazioni legali e la licenza d'uso per il motore di raytracing Persistence of Vision™ chiamato anche POV-Ray™.
Prima di usare questo programma si devono leggere i paragrafi seguenti.

Appendice A .1 Accordo Sulla Licenza

QUESTO MESSAGGIO DEVE ESSERE INCLUSO IN TUTTI I FILE DI PERSISTENCE OF VISION, UFFICIALI O PERSONALIZZATI. NON PUÒ ESSERE CANCELLATO O MODIFICATO. LE INFORMAZIONI IN ESSO INCLUSE TRATTANO DELL'UTILIZZO DEL PACCHETTO SOFTWARE IN TUTTO IL MONDO. IL DOCUMENTO SOSTITUISCE OGNI PRECEDENTE LICENZA D'USO O PERMESSO DI DISTRIBUZIONE. OGNI INDIVIDUO, COMPAGNIA O GRUPPO CUI SONO STATE CONCESSE LICENZE SPECIALI POSSONO CONTINUARE A DISTRIBUIRE LA VERSIONE 2.x MA DEVONO RICHIEDERLE PER LE VERSIONI 3.00 O SUCCESSIVE.

Questo documento riguarda l'uso e la distribuzione del motore di raytracing Persistence of Vision™ chiamato anche POV-Ray™. Esso si applica a tutti i codici sorgenti del programma, ai file eseguibili (binari), ai file di descrizione delle scene, file contenenti la documentazione, file di aiuto, immagini e file .INI contenuti negli archivi ufficiali distribuiti dal POV-Ray Team™. A tutto questo ci si riferirà come al 'software'.

Tutto questo software è posto sotto diritto di riproduzione (copyright 1991,1997) da parte del POV-Ray Team™. Sebbene sia distribuito come software gratuito, NON è di pubblico dominio.

Il pacchetto software può SOLO essere distribuito e/o modificato in accordo alla licenza qui espressa. Lo scopo della licenza è di promuovere POV-Ray come un programma di raytracing standard, fornire l'intero pacchetto software gratuitamente a quante più persone sia possibile, evitare lo sfruttamento di utilizzatori e sviluppatori di POV-Ray, migliorare la qualità della vita di coloro che interagiscono con POV-Ray. Questa licenza è stata creata in modo che questi scopi potessero essere raggiunti. Chi legge è legalmente tenuto a seguire queste regole, ma speriamo che esse vengano seguite più per principio di etica, che non per evitare controversie.

Appendice A .2 Permesso di Uso

Si permette all'utente di utilizzare il software ed i file ad esso associati in questo pacchetto per creare e renderizzare immagini. L'utilizzo del software allo scopo di creare immagini è completamente gratuito. Il creatore di una scena e dell'immagine ottenuta da essa, detiene ogni

diritto sulla scena e sull'immagine e può utilizzarli per ogni scopo, commerciale e non.
All'utente viene anche garantito il diritto di utilizzare nelle proprie scene i file di descrizione delle scene, caratteri, immagini e file 'include' nelle directory **include**, **texsamps**, **pov3demo**. Questo diritto non si estende ai file contenuti nella directory **povscn**. Questi ultimi sono presenti a scopo di svago ed educativo, ma non possono essere usati come base di alcun lavoro.

Appendice A .3 Regole Generali per Tutte le Distribuzioni
Si permette in anticipo di distribuire questo pacchetto software sotto certe condizioni specifiche, purché vengano rispettate le seguenti condizioni.

Questi archivi non devono essere ri-archiviati usando un metodo diverso senza l'esplicito permesso del POV-Ray Team. Gli archivi possono essere rinominati solo per esigenze di sistema, o per evitare nomi duplicati, ma chiediamo che si conservino i nomi originali per quanto sia possibile (ad esempio, **povsrc.zip** e **povsrc30.zip**)

La distribuzione su CD-ROM di una versione decompressa e funzionante del programma è permessa se i file sono disposti nella nostra struttura di directory come se fossero stati correttamente installati su di un hard disk.

Si deve distribuire un pacchetto completo come viene descritto nel prossimo paragrafo. Nessuna parte di questo pacchetto può esserne separata e distribuita separatamente se non sotto le condizioni specificate più avanti.

La distribuzione non commerciale su cui non viene posto un compenso di qualunque natura (ad esempio, un utente che copia il software per un amico o collega) è permessa senza altre restrizioni.

Insegnanti ed istituzioni didattiche possono distribuire il software ai loro studenti facendo pagare un costo minimo per la copia, se il software deve essere usato all'interno di un corso.

Appendice A .4 Definizione di Pacchetto Completo
POV-Ray è contenuto in due archivi separati per ogni piattaforma per la quale viene distribuito. Un pacchetto software completo consiste in :

Archivi eseguibili direttamente, contenenti un programma eseguibile, la documentazione e le scene di esempio, ma non il codice sorgente

oppure

Archivi destinati al programmatore, contenenti il codice sorgente del programma competo, ma non i file eseguibili. Si deve includere un archivio contenente le scene di esempio e la documentazione. Per alcune piattaforme, la documentazione e le scene campione sono archiviate separatamente dal codice sorgente. Il codice sorgente da solo non basta. Devono esserci anche documentazione e scene d'esempio.
POV-Ray è distribuito nelle versioni ufficiali per MS-Dos, Windows 32, Linux per processori serie Intel x86, Apple Macintosh, Apple PowerPC, SunOS ed Amiga. Altri sistemi possono essere aggiunti in futuro.

Chi distribuisce POV-Ray non è tenuto a offrire supporto per tutte le piattaforme, ma per ogni piattaforma supportata deve essere resa disponibile una versione completa del programma. Ad esempio, una BBS dedicata al Macintosh non è obbligata a distribuire la versione per Windows.

Il software può essere associato ad altri pacchetti software solo alle condizioni specificate nelle istruzioni seguenti.

Appendice A .5 Condizioni per le Compagnie di Distribuzione di Programmi Shareware / Freeware

Le compagnie di distribuzione di programmi 'shareware' e 'freeware' possono distribuire il programma in raccolte di solo software utilizzando mezzi come (ma non solo) floppy disk, CD-ROM, nastro magnetico, dischi ottici, hard disk, o schede di memoria. Questa sezione si applica solo ai distributori di raccolte di programmi. Chiunque voglia unire il programma ad un prodotto 'shareware' deve utilizzare le regole commerciali. Ogni associazione del programma con libri, giornali, od altri media stampati dovrebbe avvalersi delle regole di distribuzione commerciali.
Si deve notificare al POV-Ray Team che si sta distribuendo POV-Ray e fornirci informazioni su come contattare il distributore, qualora dovessero verificarsi controversie.

Non possono venire ricaricati più di cinque dollari USA ($5) per ogni dischetto sul quale viene copiato il software, compreso il prezzo del dischetto. Lo spazio su ogni dischetto deve essere utilizzato al massimo. Non si possono dividere i file su più dischi di quanto sia necessario.

La distribuzione su supporti ad alta capacità, come nastro magnetico o CD-ROM è permessa se il costo finale per l'utente è inferiore a $0.08 dollari per megabyte di dati. Ad esempio, un CD-ROM con 600 megabyte di dati non può costare più di 48 dollari.

Appendice A .6 Condizioni per Servizi On Line e BBS incluso Internet.

Servizi on-line, BBS e siti Internet possono distribuire il software sotto le condizioni elencate in questa sezione. I siti che permettono agli utenti di eseguire POV-Ray da posizioni remote devono attenersi alle condizioni specificate nel paragrafo seguente.
Gli archivi devono essere facilmente disponibili e raccolti in un'area (serie di directory) con programmi simili.
Si richiede che i siti di distribuzione on-line rimuovano le precedenti versioni di POV-Ray per evitare confusione all'utente e semplificare o minimizzare i nostri sforzi di supporto.

Il sito può ricaricare solo tariffe di utilizzo standard per il downloading del software. Non si possono richiedere extra per questo pacchetto. Ad esempio, CompuServe o America On Line possono rendere disponibile il software ai loro utenti, ma possono richiedere solo le normali tariffe di uso per il tempo necessario al downloading del software.

Appendice A .7 Esecuzione remota di POV-Ray

Alcuni siti internet sono stati impostati in maniera tale che utenti remoti possano eseguire POV-Ray sul server internet. Altre società vendono tempo di CPU per eseguire POV-Ray su workstation o computer ad alta velocità. Un simile utilizzo di POV-Ray è permesso alle seguenti condizioni :

Tariffe per simili servizi, devono venire imposte ESCLUSIVAMENTE sulla base del tempo di connessione, dello spazio disco utilizzato e dell'utilizzo del processore. Non possono venire ricaricate tariffe per l'utilizzo di POV-Ray oltre a quelle imposte per l'utilizzo di altro software. Si devono informare chiaramente gli utenti che le tariffe che pagano sono per l'utilizzo del computer e non per l'utilizzo del software POV-Ray.

Gli utenti devono essere chiaramente informati che stanno usando POV-Ray, che il software è gratuito e dove possono reperire le versioni ufficiali del programma. Qualunque tentativo di oscurare il fatto che l'utente sta utilizzando POV-Ray è espressamente proibito.

Tutti i file normalmente disponibili in una distribuzione completa e soprattutto una copia di questa licenza e la documentazioni devono essere resi disponibili per downloading o leggibili on line in modo che gli utilizzatori del programma abbiano accesso al materiale del pacchetto completo.

Appendice A .8 Condizioni per la Distribuzione di Versioni Personalizzate.

L'utente ha il diritto e il privilegio di poter modificare e compilare il codice sorgente per il proprio uso personale in qualunque modo desideri. 'Ciò che fai in casa tua col software sono affari tuoi'.

Se l'utente vuole distribuire una versione modificata del software, della documentazione o di altre parti del pacchetto deve seguire le istruzioni fornite qui. Ciò comprende ogni traduzione del manuale in altri linguaggi o altri formati. Queste istruzioni sono state scritte per promuovere la crescita di POV-Ray e prevenire le difficoltà per gli utenti e per gli sviluppatori del programma. Qualora si creasse una versione personalizzata, seguirle attentamente per il beneficio di tutte le persone coinvolte.

Nessuna porzione del codice sorgente di POV-Ray può essere inclusa in un altro programma a meno che non sia chiaramente una versione personalizzata di POV-Ray che include tutte le funzioni elementari di POV-Ray..

Tutti gli eseguibili, la documentazione, i file modificati e le descrizioni degli stessi devono chiaramente identificarli come una versione modificata e non ufficiale del programma. Qualunque tentativo di nascondere il fatto che l'utente sta utilizzando POV-Ray o che la versione che sta utilizzando non è la versione ufficiale è espressamente proibito.

Pieno supporto deve essere fornito a tutti coloro che stanno utilizzando la versione personalizzata. Devono venire fornite le informazioni necessarie per fare sì che gli utilizzatori possano contattare l'autore della versione personalizzata. IL POV-Ray Team non è obbligato a fornire all'autore o ai suoi utenti alcun tipo di supporto tecnico.

Si devono includere le informazioni nelle macro DISTRIBUTION_MESSAGE nel file sorgente **optout.h** ed assicurarsi che il programma mostri queste informazioni. Si devono mostrare tutte le note di copyright e le schermate di 'credits' con i nomi degli autori della versione ufficiale. Le versioni personalizzate possono solo essere distribuite come 'freeware'. Tutte le modifiche apportate a POV-Ray devono essere disponibili pubblicamente e gratuitamente e il codice sorgente deve essere distribuito per tutte le nuove parti della versione personalizzata. Lo scopo è che gli utilizzatori possano ricompilare il programma da soli e migliorarlo ulteriormente con le loro modifiche.

Si deve fornire la documentazione per tutte le modifiche apportate al programma che viene distribuito. Devono essere incluse chiare informazioni su come ottenere le versioni ufficiali di POV-Ray.

L'utente è incoraggiato a mandare miglioramenti e correzioni del programma al POV-Ray Team, ma il Team non è in nessun modo tenuto ad utilizzarli. Mandando il materiale al Team, chi contribuisce asserisce la sua proprietà ed il diritto di distribuzione del materiale. Autorizza il Team ad utilizzarlo in qualunque modo. Chi contribuisce mantiene ancora i diritti sul materiale donato, ma con la donazione passa tutti i diritti relativi all'utilizzo ed alla distribuzione al POV-Ray Team. Il Team non è obbligato ad usare il materiale, ma in caso positivo chi contribuisce non acquisisce diritti relativi a POV-Ray. Il Team fornirà credito come creatore di nuovo codice ove applicabile.

Includere una copia del file **povlegal.doc**.

Appendice A .9 Condizioni per il Bundling Commerciale

I rivenditori che desiderino unire POV-Ray ad altro software commerciale (incluso 'shareware') o con pubblicazioni devono prima ottenere esplicita autorizzazione dal POV-Ray Team. Ciò include qualunque software commerciale o pubblicazione, come (ma non solo) riviste, libri, giornali. Il POV-Ray Team deciderà se questa distribuzione sarà autorizzata su base individuale e può imporre certe restrizioni se ne vede la necessità. I termini minimi per la distribuzione sono forniti qui. Altre condizioni possono essere imposte.

Gli acquirenti del prodotto non devono essere indotti a credere che stanno pagando per POV-Ray. Qualunque menzione di POV-Ray fatta sul contenitore, nella pubblicità o nel manuale di istruzioni deve essere fornita con la chiara informazione che POV-Ray è software gratuito e che può essere ottenuto gratis o per un costo simbolico da varie fonti.

Devono essere incluse informazioni su come ottenere la versione ufficiale di POV-Ray.

Si deve fornire pieno supporto per tutti gli utenti che sono entrati in possesso di POV-Ray attraverso il prodotto. Il POV-Ray Team non è obbligato a fornire al distributore o alla sua clientela qualunque supporto tecnico.

Si deve includere una (o più) pagine di credito per POV-Ray.

Se viene modificata una qualunque parte di POV-Ray per l'utilizzo con hardware o software particolari, si devono seguire anche le norme per la distribuzione di versioni personalizzate.

Devono essere fornite informazioni di supporto tecnico e su come contattare il distributore.

Si deve includere un pacchetto competo come viene definito nei paragrafi precedenti.

Appendice A .10 Valore Commerciale del Software

Sebbene POV-Ray sia, quando distribuito entro i termini di questa licenza, gratuito, il valore commerciale (o prezzo) di questo software è determinato in 20 dollari USA per ogni copia distribuita, copiata o usata. Se il software è distribuito copiato o usato senza autorizzazione, si è legalmente in debito col detentore del copyright o con qualunque persona od organizzazione delegata da esso per la riscossione di questo debito. Il debito deve essere saldato entro 30 giorni dall'evento.

Comunque, nessuno dei paragrafi precedenti costituisce autorizzazione a distribuire, copiare o usare questo software al di fuori dei termini della licenza. In particolare le condizioni ed il valore del software (che sia stato pagato o no) non eliminano le penalità previste dalla legge e non costituiscono una soluzione che permetterebbe di evitare altri rimedi ai termini della legge che possono essere disponibili al detentore del copyright.

In poche parole : POV-Ray è gratis se stai alle nostre condizioni, altrimenti no. Il detentore del copyright sceglie di fornirlo gratis solo ed esclusivamente a queste condizioni.

Per le regole di copyright il valore del software è fissato a 20 dollari. Nel caso di esecuzione del programma on line o remota, ogni istanza di esecuzione è considerata come una distribuzione del software.

Appendice A .11 Altri permessi

Il Team permette ed incoraggia la creazione di altri programmi, anche commerciali che importino, esportino o traducano file nel linguaggio di descrizione delle scene di POV-Ray. Non ci sono limitazioni all'uso del linguaggio in sé. CI riserviamo il diritto di modificare qualunque parte del linguaggio.

"POV-Ray", "Persistence of Vision", "POV-Ray Team" e "POV-Help" sono marchi registrati dal POV-Ray Team.

Mentre non viene fatta alcuna restrizione sulle lettere "POV", si richiede di non utilizzare "POV" nel nome di altri prodotti. Ciò tende ad asserire implicitamente che il prodotto appartiene al POV-Ray Team. I creatori di programmi esistenti che utilizzano "POV" precedentemente alla pubblicazione di questo documento non devono ritenersi colpevoli, posto che chiariscano che il loro programma non è un prodotto del POV-Ray Team né viene supportato da esso.

Appendice A .12 Revoca della Licenza.

LA VIOLAZIONE DI QUESTA LICENZA COSTITUISCE UN'INFRAZIONE ALLE LEGGI SUL COPYRIGHT. RISULTERA' NELLA REVOCA DI TUTTI I PRIVILEGI SULLA DISTRIBUZIONE E PUO' RISOLVERSI IN PENE CIVILI O PENALI.

Coloro che violano questa licenza e a cui viene proibita la distribuzione del programma saranno identificati in questo documento.

A tale proposito, "PC Format", una rivista pubblicata da Future Publishing, Ltd. nel Regno Unito, ha distribuito versioni incomplete di POV-Ray 1.0 in violazione alla licenza valida in quel periodo. In seguito hanno cercato di distribuire la versione 2.2 del programma in violazione alla licenza valida in quel periodo. Ci sono prove che altre società del gruppo "Future Publishing" hanno violato i nostri termini. Pertanto, si proibisce a "PC Format" ed a qualunque altra rivista, libro o pubblicazione in CD-ROM di proprietà del gruppo "Future Publishing" la distribuzione del software POV-Ray fino a nuovo ordine.

Appendice A .13 Scarico di Responsabilità.

Il software è fornito come è senza qualunque garanzia. Sebbene gli autori abbiano cercato di scoprire e correggere qualunque difetto nel software, non sono responsabili per qualunque danno o perdita di qualunque tipo causata dall'uso o dall'abuso del software. Gli autori non sono obbligati a fornire servizio, correzioni o miglioramenti a questo pacchetto.

Appendice A .14 Supporto Tecnico.

Speriamo sinceramente che vi divertiate col nostro programma. Se avete problemi col programma il Team vorrebbe esserne informato. Se ci sono commenti, domande o possibili miglioramenti, contattate il POV-Ray Team sul forum POVRAY di CompuServe. Visitate anche i siti internet *http://www.povray.org* ed *ftp.povray.org*. Il gruppo USENET *comp.graphics.rendering.raytracing* è una grande fonte di informazioni su POV-Ray e sugli argomenti correlati.
Richieste di licenza e di limitato supporto tecnico dovrebbero essere fatte via e-mail a :

Chris Young
POV-Ray Team Coordinator
CIS: 76702,1655
Internet 76702.1655@compuserve.com

Il seguente indirizzo postale è solo per questioni di licenza, qualora non fosse possibile inviare posta elettronica.

Non forniamo supporto per posta, solo per posta elettronica (e non ci interessa se non hai un modem o accesso a servizi on line). Non spediamo dischi con aggiornamenti. Non spedite soldi.

Chris Young
3119 Cossell Drive
Indianapolis, IN 46224 U.S.A.

Altre informazioni sugli autori possono essere trovate nei paragrafi dell'appendice "Autori" (Appendice B) e "Cartoline per i Membri del POV-Ray team" (Appendice D).

Appendice B Autori

Segue una lista in ordine alfabetico degli autori che hanno lavorato nel POV-Ray Team o che hanno partecipato in maniera rilevante. Se vuoi contattarli o ringraziarli leggi i paragrafi "Contattare gli Autori" (Appendice C) e "Cartoline per i Membri del POV-Ray team" (Appendice D).

Claire Amundsen
(Tutorial per il manuale di POV-Ray)

Steve Anger
(programmatore delle versioni 2.0/3.0 di POV-Ray)
CIS: 70714,3113
Internet: sanger@hookup.net

Randy Antler
(miglioramenti al codice per il display su personal IBM)

John Baily
(codice per immagini Targa RLE)

Eric Barish
(codice per la nebbia rasoterra)

Dieter Bayer
(programmatore di POV-Ray 3.0 e coordinatore della documentazione)
CIS: 104707,643

Kendall Bennett
(supporto per la libreria PMODE)

Steve Bennett
(supporto GIF)

Jeff Bowermaster
(Beta tester)

David Buck
(autore di DKBTrace)
(programmatore di POV-Ray 1.0)

Chris Cason
(programmatore di POV-Ray 2.0/3.0, POV-Help, POV-Ray per Windows)

Internet (preferibilmente): povray@mail.oaks.com.au or Chris.Cason@povray.org
CIS: 104706,3166

Aaron Collins
(co-autore di DKBTrace 2.12)
(programmatore di POV-Ray 1.0)

Chris Dailey
(programmatore di POV-Ray 3.0)
CIS:

Steve Demlow
(programmatore di POV-Ray 3.0)
CIS:

Andreas Dilger
(programmatore di POV-Ray 3.0)
Internet: adilger@enel.ucalgary.ca
WWW: http://www-mddsp.enel.ucalgary.ca/People/adilger/

Joris van Drunen Littel
(beta tester per la versione Mac)

Alexander Enzmann
(programmatore di POV-Ray 1.0/2.0/3.0)
CIS: 70323,2461
Internet: xander@mitre.com

Dan Farmer
(programmatore di POV-Ray 1.0/2.0/3.0)
CIS: 74431,1075

Charles Fusner
(Tutorial per il manuale di POV-Ray)

David Harr
(aiuto a vignetta per Mac e codice per la tavolozza)

Jimmy Hoeks
(File di Aiuto per l'interfaccia di Windows)

Terry Kanakis
(Miglioramenti per la Macchina Fotografica)

Kari Juharvi Kivisalo
(codice per la nebbia rasoterra)

Adam Knight
(beta tester per la versione Mac, sviluppatore della guida per Mac)
CIS:

Lutz Kretzschmar
(codice per il display su IBM-PC [SS24 truecolor], parte del codice per l'anti-aliasing)
CIS: 100023,2006

Charles Marslett
(codice per il display su IBM-PC)

Pascal Massimino
(Frattali)

Jim McElhiney
(programmatore di POV-Ray 3.0)
CIS:

Robert A. Mickelsen
(manuale di POV-Ray 3.0)
internet email: ram@iu.net
compuserve: 71042,751
web: www.websharx.com/~kahuna

Mike Miller
(Artista, scene, stones.inc)
CIS: 70353,100

Douglas Muir
(mappe bump, height field)

Joel NewKirk
(versione per Amiga)
CIS: 102627,1152

Jim Nitchals
(file scena, versione Mac)

Paul Novak
(Texture)

Dave Park
(supporto per Amiga, codice video AGA)

David Payne
(codice per Targa RLE)

Bill Pulver
(codice per il tempo, compilazione su IBM-PC)

Anton Raves
(Beta tester, ha aiutato su diverse cosette relative al Mac)
CIS: 100022,2603

Dan Richardson

(Manuali)
CIS:

Tim Rowley
(supporto ai formati PPM e BMP)
Internet: trowley@geom.umn.edu

Robert Schadewald
(Beta tester)
CIS:

Eduard Schwan
(Versione Mac, anteprima a mosaico, manuale)
CIS: 104706,3276 or POVRAYMAC or ESPSW
Internet: povraymac@compuserve.com or espsw@compuserve.com
WWW: http://ourworld.compuserve.com/homepages/povraymac

Robert Skinner
(funzioni Noise)

Erkki Sondergaard
(File Scena)
CIS:

Zsolt Szalavari
(Codice per gli Aloni)
Internet: zsolt@cg.tuwien.ac.at

Scott Taylor
(texture Leopard e onion)

Timothy Wegner
(Oggetti frattali, supporto per il formato PNG)
CIS: 71320,675
Internet: twegner@phoenix.net

Drew Wells
(programmatore di POV-Ray 1.0, coordinatore del POV-Ray 1.0 Team)

Chris Young
(programmatore di POV-Ray 1.0/2.0/3.0, coordinatore del POV-Ray 2.0/3.0 Team)
CIS: 76702,1655

Appendice C Contattare gli Autori

Il POV-Ray Team è una squadra di programmatori, progettisti, animatori ed artisti volontari che si incontrano via posta elettronica sul forum POVRAY di CompuServe.
Lo scopo del POV-Ray Team è di creare software di alta qualità per il rendering e l'animazione scritto in C che possa essere portato su molte piattaforme.
Se hai domande su POV-Ray contatta :

Chris Young

Coordinatore del POV-Team
CIS: 76702,1655
Internet: 76702.1655@compuserve.com

Amiamo sapere come state usando il programma. Faremo del nostro meglio per cercare di risolvere qualunque problema che possiate avere con POV-Ray ed aggiungere buoni suggerimenti all'interno del software.
Se ci sono domande riguardanti la distribuzione, l'utilizzo commerciale, o qualunque argomento particolarmente difficoltoso , contattate Chris Young, coordinatore del Team di sviluppo. Altrimenti spargete la posta in giro. A tutti noi piace avere vostre notizie !

Il miglior metodo di contattarci è la posta elettronica attraverso CompuServe per la maggior parte di noi. Adesso anche America On-Line e Internet possono mandare posta a CompuServe, quindi usando l'indirizzo Internet la posta ci sarà inoltrata tramite CompuServe dove la leggiamo ogni giorno.

Per favore, non mandate grossi file attraverso la posta senza chiedere prima. Paghiamo per ogni minuto su CompuServe e file di grandi dimensioni possono diventare costosi. Chiedete permesso prima di mandarci il file, grazie !

Appendice D Cartoline per i Membri del POV-Ray Team

Se vuoi ringraziare personalmente alcuni membri del POV-Ray Team puoi spedirgli una cartolina da qualunque posto ti trovi. Per evitare indirizzi non validi, nel caso che qualcuno di noi traslochi, gli indirizzi forniti sotto (in ordine alfabetico) sono validi solo fino alla data indicata

Dieter Bayer
Taeublingstr. 26
91058 Erlangen
Germany (fino al 31/7/1997)

Chris Cason (versione Windows)
PO Box 407
Williamstown
Victoria 3016
Australia (fino al 31/12/1998)

Joel NewKirk
255-9 Echelon Rd
Voorhees, NJ, USA, 08043

Eduard Schwan (versione Macintosh)
1112 Oceanic Drive
Encinitas, California, USA, 92024-4007 (fino al 30/6/1998)

Appendice E Ringraziamenti

Ringraziamenti per avere contribuito al manuale del programma vanno a (in ordine alfabetico) :

Charles Fusner (tutorial su blob, lathe e prisma)

Appendice F Suggerimenti e Consigli

Appendice F .1 Suggerimenti per il Design delle Scene

Esiste una grande quantità di ottimi modellatori shareware disponibili per DOS che creeranno file scena di POV-Ray. Le fondi elencate in altre parti di questo manuale sono i posti migliori dove reperirli.

Centinaia di utilità speciali sono state scritte per POV-Ray : programmi di conversione dati, generatori di oggetti, interfacce e altro. Non sarebbe possibile elencarli tutti qui, ma ancora, le fonti elencate ne avranno la maggior parte. La maggior parte, seguendo lo spirito di POV-Ray sono gratuiti, o shareware a basso prezzo.

Alcune scene estremamente elaborate sono state prima progettate su carta. Mike Miller raccomanda la carta millimetrata con divisione in decimi, che permette la conversione alle unità decimali in modo naturale.

Iniziare con una scena elementare, come una copia di **basicvue.pov** ed ampliarla. In generale, posizionare gli oggetti vicino all'origine, con la macchina fotografica rivolta nella direzione delle z negative, verso l'origine. Naturalmente questa non è una regola, ma all'inizio mantiene le cose semplici.

Per un illuminazione semplice, posiziona una sorgente luminosa vicino all'origine. Gli oggetti sembreranno piatti, ma almeno saranno visibili. Da lì, la luce può essere spostata in una posizione migliore.

Appendice F .2 Suggerimenti per la Correzione delle Scene.

Per vedere velocemente la tua immagine, renderizzala molto piccola. Con meno punti da calcolare il programma può finire più velocemente. Una dimensione di 160x100 è buona.

Usa il parametro `+q` quando è utile.

Se c'è una particolare area della scena che devi controllare ad alta risoluzione, magari con anti-aliasing (ad esempio, una texture 'legno' molto fine) , usa i parametri `+sc`, `+ec`, `+sr`, `+er` per isolare e renderizzare un riquadro.

Se la tua immagine contiene molte riflessioni reciproche, imposta `max_trace_level` ad un valore come 1 o 2. Non dimenticare di riportarlo al valore normale quando passi al rendering definitivo !

Elimina le luci non necessarie. Commenta tutte le luci estese quando non sono necessarie per la correzione della scena. Di nuovo, non dimenticare di rimetterle a posto prima di andare a letto con il rendering definitivo in esecuzione !

Se sei incorso in un errore che non riesci a vedere, è il momento di iniziare a commentare il file. Usa i caratteri `/*...*/` per eliminare la maggior parte della scena e renderizza di nuovo. Se non ci sono più errori il problema si trova ovviamente da qualche parte nella zona commentata. Un controllo lento e metodico di questo tipo porta a trovare l'errore o a impazzire serenamente. Forse ad entrambe le cose.

Se ti sembra di esserti perso o di aver perso un oggetto (succede, all'inizio) ci sono alcuni trucchi che talvolta possono aiutare:

Sposta indietro la camera per avere una visione estesa. Ciò può non essere di aiuto con oggetti

piccoli che possono essere meno visibili da lontano, ma è un aiuto.
Prova ad impostare la luce ambiente ad 1.0 se sospetti che gli oggetti siano semplicemente in ombra. Ciò li renderà luminescenti e potrai vederli anche senza luci nella scena.
Cambia l'oggetto con un sostituto più grande come una grossa sfera o un cubo. Assicurati che tutte le trasformazioni che vengono applicate all'oggetto siano applicate anche al sostituto in modo che vengano a trovarsi nella stesa posizione.

Appendice F .3 Suggerimenti per l'Animazione

Quando si animano degli oggetti con texture, la texture deve muoversi insieme all'oggetto, cioè, devi applicare le stesse rotazioni o traslazioni alla texture ed all'oggetto. Ciò viene fatto automaticamente se le trasformazioni sono applicate dopo il blocco `texture{...}` come nel seguente esempio :

```
oggetto { ...
pigment { ... }
scale < ... >
}
```

oggetto e texture verranno ridimensionati nella stessa proporzione, mentre

```
oggetto { ...
scale < ... >
pigment { ... }
}
```

ridimensionerà l'oggetto ma non la texture.
Si possono dichiarare costanti per la maggior parte dei tipi di dati usati dal programma, compresi numeri decimali e vettori. Scrivendole in un file INC separato, puoi dividere i parametri necessari per l'animazione dalla scena.
Alcuni esempi di costanti dichiarate possono essere :

```
#declare Rotazione_Y = 5.0 * clock
#declare Rotazione_Oggetto= <0, Y_Rotation, 0>
#declare La_Mia_Sfera = sphere { <0, 0, 0>, 1.1234 }
```

Altri esempi si possono trovare nei file scena di esempio.
Un suggerimento per gli utilizzatori di MS-Dos : procuratevi **DTA** (Dave's Targa Animator) per creare animazioni FLI e FLC. Programmi come **aaplay.exe** e **play.exe** sono diffusi programmi per visualizzare queste animazioni.

Quando si muove la macchina fotografica in un'animazione (o la si posiziona per una scena ferma), si deve evitare di metterla direttamente sopra l'origine, poiché ciò causa errori molto strani. E' preferibile spostarla leggermente fuori centro e evitare che si trovi ad inquadrare direttamente la scena lungo la verticale.

Appendice F .4 Suggerimenti per le Texture.

Wood (legno) è concepito come un ceppo con gli anelli di crescita allineati lungo l'asse z. In generale avranno un aspetto migliore quando saranno stati rimpiccioliti ad un decimo della dimensione originaria. Conviene iniziare con valori piccoli della turbolenza (circa 0.05 è un buon valore di inizio).

La texture Marble (marmo) è costruita come una primitiva di pigmento molto simile a `gradient x`. Quando si aggiunge turbolenza, l'effetto è diverso se osservato da un lato o dall'estremità. Prova a ruotarla di `90*y` per vedere la differenza.

Non si possono ottenere riflessi speculari su oggetti completamente neri. Si può invece provare ad utilizzare un grigio molto scuro (ad esempio, `Gray10` oppure `Gray15` in **colors.inc**).

Appendice F .5 Suggerimenti per gli Height Field

Prova ad usare POV-Ray per creare immagini di partenza per costruire height field :

```
camera { location <0, 0, -2> }
plane { z, 0
finish { ambient 1 } // non c'è bisogno di luci
pigment { bozo } // né di altro. Provare.
}
```

è tutto quello di cui hai bisogno per ottenere un file TGA che può essere usato per un height field in un'altra immagine !

Appendice F .6 Convertire la Chiralità

Se importi oggetti da altri sistemi, può capitare che essi siano rovesciati (invertiti da sinistra a destra) e che nessuna rotazione possa correggerli.
Spesso, tutto quello che c'è da fare è cambiare di segno i termini del vettore `right` della macchina fotografica, per invertirla (ed usare un sistema di coordinate destrorso). Alcuni programmi sembrano interpretare i sistemi di coordinate in maniera diversa, quindi può essere necessario sperimentare altre soluzioni se vuoi che i vettori y e z funzionino come in POV-Ray.

Appendice G Domande Frequentemente Poste (FAQ)

Questa è una raccolta di domande frequentemente poste e delle relative risposte prese direttamente dai messaggi apparsi sul forum POVRAY di CompuServe e sul newsgroup *comp.graphics.rendering.raytracing*.
Questa raccolta è rivolta soprattutto agli utenti della versione di POV-Ray per PC IBM e compatibili. Speriamo che versioni future eliminino questa tendenza ma, al momento, questo è il pubblico più vasto.

Appendice G .1 Domande Generali

Appendice G .2 Domande sulle Opzioni di POV-Ray

Domanda : Come posso impostare l'anteprima a mosaico in modo che passi da 8x8 al rendering finale senza passare da 4x4 e 2x2 ?
Risposta : Usa il parametro `+SPn` o l'opzione `Preview_Start_Size` per impostare la risoluzione iniziale e `+Epn` o `Preview_End_Size` per impostare la risoluzione finale. Con `+SP8` e `+EP8` passrà da 8x8 (in un solo passaggio) al rendering definitivo ad 1x1.

D : Conviene usare il parametro `+MB` in scene molto piccole, cioè con un piccolo numero di oggetti ?
R : Dipende dal numero e dal tipo di oggetti. Normalmente non disturba usare sempre la gerarchia di bounding (`+MB0`). Se però hai uno o due oggetti può essere meglio non usarlo.

D : Il parametro `+MB` influisce sulla qualità dell'immagine ?

R : No, influisce solo sulla velocità del rendering.

D : Come posso evitare che la schermata di testo scorra quando ricevo dei messaggi di errore ?
R : Usa il parametro `+P`. Tutte le volte che POV-Ray viene interrotto o che il rendering finisce, puoi usare i tasti cursore per rileggere il testo.

Appendice G .3 Domande sui File INC

D : Nel file **textures.v2** le texture relative al vetro sono commentate. Perché ?
R : Queste texture sono riprodotte in **glass.inc**. I vecchi nomi delle texture non adottavano il nuovo schema di denominazione delle texture. `Glass` si chiama `T_Glass1`, `Glass2` è ora `T_Glass2` e `Glass3` è `T_Glass3`. Si possono togliere i commenti ai vecchi nomi, ma vi consigliamo di usare i nuovi.

Appendice G .4 Domande sugli Oggetti

Appendice G 4.1 Domande sugli Height Field

D : Ho 8 megabyte di RAM sul mio computer e ho finito la memoria cercando di usare un file .POT di 1024x768 ed una GIF della stesa risoluzione per due scene diverse. Sono io che ho poca memoria o è da pazzi usare una risoluzione del genere ?
R : Gli height field smussati consumeranno molta più memoria (circa tre volte tanto) di quelli non smussati. Dato che smussare non è necessario a queste risoluzioni non farlo.

D : Voglio usare la stessa immagine per una mappa e per l'height field (ottenendo in questo modo una relazione tra la quota ed il colore). Funziona solo quando uso GIF o BMP ? Non sono riuscito a mappare il formato POT.
R : I file POT non sono supportati per la mappatura delle immagini. Il loro scopo in POV-Ray è di essere usati come base per un height field. Questo non impedisce però di ottenere da **fractint** un'immagine GIF che usa i colori che hai usato per il file POT ed usarla come una mappa. Ricorda però che mentre gli height field si trovano sul piano x-z la mappa giace sul piano x-y,quindi dovray ruotare la mappa di 90° attorno all'asse x (usa `rotate x*90`) per fare allineare la mappa all'height field.

Appendice G 4.2 Domande sul Testo

D : E' possibile sapere in qualche modo la dimensione di un oggetto testo ?
R : Purtroppo no. Ci servirebbe veramente, ma non è facile. Fino a pochi giorni prima dell'uscita della versione beta non funzionava nemmeno la spaziatura orizzontale. Ora che l'abbiamo aggiustata, forse troveremo un modo per leggere la lunghezza del testo.

Appendice G .5 Domande Atmosferiche

Appendice G .5.1 Domande sull'Atmosfera

D : Perché l'atmosfera che ho aggiunto non è visibile ?
R : l'errore più comune è non aggiungere la parola chiave `hollow` a tutti gli oggetti entro cui si trova la macchina fotografica. Se sei all'interno di un parallelepipedo che serve per modellare una stanza, devi aggiungere la parola chiave `hollow` alla frase `box{...}`, se ti servi di un piano per modellare il suolo dovrai aggiungergli `hollow` (solo se ti trovi all'interno del piano, ma per esserne sicuro, puoi farlo sempre).

Se questo non funziona possono esserci altri problemi da verificare. I valori di `distance` e `scattering` devono essere maggiori di zero. Le sorgenti luminose che devono interagire con

l'atmosfera non devono contenere il comando `atmosphere off`.

D : Perché non vedo l'atmosfera attraverso un oggetto trasparente ?
R : Se hai un oggetto trasparente, devi renderlo quasi sempre cavo aggiungendo la parola chiave `hollow`. Tutte le volte che viene rilevata un intersezione ed il raggio cade all'interno di un oggetto solido, l'atmosfera non viene calcolata.

Ad esempio, se hai un piano parzialmente trasparente, l'atmosfera dall'altra parte del piano sarà visibile solo se il piano sarà reso cavo usando `hollow`.

D : Perché le parti illuminate dell'atmosfera rischiarano lo sfondo ?
R : Per prima cosa, non lo fanno.
Secondo, quando zone dello sfondo sono visibili attraverso l'atmosfera e le corrispondenti parti dell'atmosfera sono illuminate da una qualsiasi sorgente luminosa, la luce dispersa è sommata alla luce che proviene dallo sfondo. Questo è il motivo per cui lo sfondo sembra essere schiarito dall'atmosfera.
Immagina l'esempio seguente : hai uno sfondo blu, attenuato dall'atmosfera in modo che il colore che raggiunge l'osservatore sia <0, 0, 0.2>. Una luce che giunge da una sorgente è attenuata e dispersa dall'atmosfera e raggiunge infine l'osservatore con un valore <0.5, 0.5, 0.5>. Dato che abbiamo già la luce proveniente dallo sfondo, i due colori si sommano per dare <0.5, 0.5, 0.7>. Quindi la luce risulta avere una sfumatura blu. Ne risulta che pensi che lo sfondo sia schiarito, ma ciò non avviene, come la seguente scena mostra chiaramente.

```
#version 3.0
camera {
location <0, 6, -20>
look_at <0, 6, 0>
angle 48
}

atmosphere {
type 1
samples 10
distance 20
scattering 0.3
aa_level 3
aa_threshold 0.1
jitter 0.2
}

light_source { <0, 15, 0> color rgb .7 shadowless }

light_source {
<-5, 15, 0> color rgb <1, 0, 0>
spotlight
point_at <-5, 0, 0>
radius 10
falloff 15
tightness 1
atmospheric_attenuation on
}
```

```
light_source {
<0, 15, 0> color rgb <0, 1, 0>
spotlight
point_at <0, 0, 0>
radius 10
falloff 15
tightness 1
atmospheric_attenuation on
}

light_source {
<5, 15, 0> color rgb <0, 0, 1>
spotlight
point_at <5, 0, 0>
radius 10
falloff 15
tightness 1
atmospheric_attenuation on
}

plane { z, 10
pigment { checker color rgb<1, 0, 0> color rgb<0, 1, 0> }
hollow
}
```

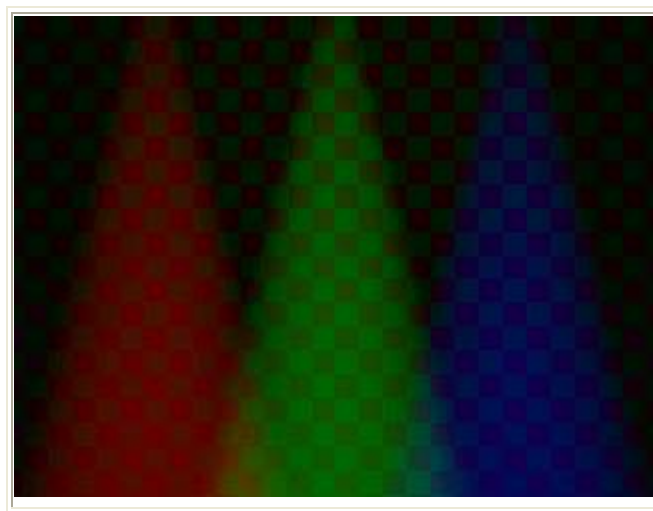



Fig. 233-"Effetti atmosferici"

Nello sfondo si vede un piano a scacchi rossi e verdi. Il colore dello sfondo visibile attraverso l'atmosfera è aggiunto alla luce dispersa degli spot. Noterai che, nonostante i quadrati rossi dietro lo spot rosso siano più chiari di quelli non illuminati da esso, quelli verdi non lo sono. Per lo spot verde, la situazione è capovolta : i quadrati verdi sembrano essere amplificati mentre i rossi no. Lo spot blu non mostra questo effetto.

D : Il manuale dice che il parametro `distance` dell'atmosfera funziona nello stesso modo di quello per la nebbia. E' vero ?
R : Sì, è vero. Prova a usare `fog` invece di `atmosphere`. Se tutto sembra a posto, in altre parole

se puoi sempre vedere lo sfondo o tutto quello che vuoi vedere, usa la stessa distanza e lo stesso colore per l'atmosfera.

Appendice G .5.2 Domande sulla Nebbia

D : Sto usando **moray** per costruire una scena che contiene della nebbia rasoterra. Il problema è che la nebbia sfuma lungo l'asse y. In **moray** l'asse delle altezze è l'asse z, quindi io ottengo un muro di nebbia, invece di uno strato orizzontale. Che fare ?
R : C'è una parola chiave che si chiama `up` e che può essere usata per specificare la direzione verticale ella nebbia. Aggiungendo la linea `up z` alla tua nebbia risolverai il problema.

Appendice H   Letture Consigliate

Oltre ai libri specifici citati nel paragrafo "Libri e CD-ROM su POV-Ray" (vedi § 2.4.6) ci sono alcuni buoni libri o periodici che dovresti poter trovare in negozi specializzati e grandi librerie, o nelle biblioteche universitarie.

"An Introduction to Ray tracing"
Andrew S. Glassner (editor)
ISBN 0-12-286160-4
Academic Press
1989

"3D Artist" Newsletter
"The Only Newsletter about Affordable
PC 3D Tools and Techniques")
Publisher: Bill Allen
P.O. Box 4787
Santa Fe, NM 87502-4787
(505) 982-3532

"Image Synthesis: Theory and Practice"
Nadia Magnenat-Thalman and Daniel Thalmann
Springer-Verlag
1987

"The RenderMan Companion"
Steve Upstill
Addison Wesley
1989

"Graphics Gems"
Andrew S. Glassner (editor)
Academic Press
1990

"Fundamentals of Interactive Computer Graphics"
J. D. Foley and A. Van Dam
ISBN 0-201-14468-9
Addison-Wesley
1983

"Computer Graphics: Principles and Practice (2nd Ed.)"

J. D. Foley, A. van Dam, J. F. Hughes
ISBN 0-201-12110-7
Addison-Wesley,
1990

"Computers, Pattern, Chaos, and Beauty"
Clifford Pickover
St. Martin's Press

"SIGGRAPH Conference Proceedings"
Association for Computing Machinery
Special Interest Group on Computer Graphics

"IEEE Computer Graphics and Applications"
The Computer Society
10662, Los Vaqueros Circle
Los Alamitos, CA 90720

"The CRC Handbook of Mathematical Curves and Surfaces"
David von Seggern
CRC Press
1990

"The CRC Handbook of Standard Mathematical Tables"
CRC Press
The Beginning of Time

Indice analitico

I numeri che seguono il segno "§" indicano il paragrafo in cui occorre la voce dell'indice analitico. Tali richiami sono comunque collegati ai rispettivi paragrafi. Le voci sono rigorosamente in ordine alfabetico, così come i numeri che richiamano i paragrafi sono nello stesso ordine in cui appaiono nella guida.Questo indice non è inclusivo di tutte le voci che avrebbero avuto "diritto" di trovarvici, ma solo delle principali.

Indice analitico delle parole chiave

	tga thickness threshold tightness tile2 tiles torus track transform translate transmit triangle triangle_wave true ttf turbulence turb_depth type
U	u ultra_wide_angle union up use_color use_colour use_index u_steps
V	v val variance vaxis_rotate vcross vdot version vlength vnormalize volume_object volume_rendered vol_with_light vrotate v_steps
W	warning warp water_level waves while width wood wrinkles
X	x

Y	y yes
Z	z
#	#break #case #debug #declare #default #else #end #error #if #ifdef #ifndef #include #max_intersections #max_trace_level #range #render #statistics #switch #version #warning #while

Manuale di Utilizzo di POV-Ray™ - Traduzione Italiana

Il "Gruppo di Lavoro POV.IT" :

Coord. progetto, ottimizzazione, impaginazione, versione POSTSCRIPT™, immagini

Dario Agosta (Firenze)

(agosta@freemail.it)

Conversione, impaginazione, versione HTML, immagini

Laura Nutini (Firenze)

(nutini@freemail.it)

Traduzione a cura di:(in ordine alfabetico):

Dario Agosta (Firenze)

(agosta@freemail.it)

capitoli 1 - 4.10.6.5, 7.5.1.1 - 7.5.5.7 e 7.6.4.5.1 - 7.8.9.3 (più appendici)

Laura Nutini (Firenze)

(nutini@freemail.it)

capitoli 6 - 6.2.5.4, 7.5.6 - 7.6.4.3.4, 7.5.1.1 - 7.5.5.7, 7.6.4.5.1- 7.8.9.3 (più appendici)

Ha partecipato alla traduzione:

Gaetano Forte (Roma)

(md5304@mclink.it)

capitoli 7 - 7.4.3

Hanno inoltre collaborato alla traduzione:

Luca Ibba (Firenze)

(l.ibba@agora.stm.it)

capitoli 7.4.4 - 7.5.2.7

Filippo Vitale (Bologna)

(filippo@joy.it)

capitoli 7.6.4.4 - 7.6.5